Cincom

SUPRA SERVER PDM

System Administration Guide
(UNIX)

P25-0132-46

SUPRA® Server PDM System Administration Guide (UNIX)

Publication Number P25-0132-46

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 45246-3731
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: http://www.cincom.com

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

The *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132-46, is dated January 15, 2002. This document supports Release 1.3 of SUPRA Server PDM in UNIX environments.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the survey provided with the online documentation at your convenience.

Cincom Technical Support for SUPRA Server PDM

FAX: (513) 612-2000
Attn.: SUPRA Server Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn.: SUPRA Server Support
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

Contents

About this book

Using this document

This manual describes:

- SUPRA Server components and related products
- How to implement the logical names required to set up your UNIX environment to run SUPRA Server
- Allowable changes to the SUPRA Directory and how to make those changes
- How to implement the logical names and other steps required to initiate the PDM
- How to communicate with the PDM using the PDM operator interface
- Information that will help you get the best performance from your SUPRA Server database
- How to use online Help for SUPRA Server
- How to use database migration instructions

The appendixes provide sample user exits, description of PDM statistics output, and an example mailbox reading program.

Document organization

The information in this manual is organized as follows:

Chapter 1—Overview of SUPRA Server
Describes SUPRA Server, the Physical Data Manager, the directory, and MANTIS.

Chapter 2—Setting up UNIX to run SUPRA Server
Describes how to install SUPRA Server PDM, modify UNIX system parameters, define environment variables, start daemon processes, and write PDM user exits.

Chapter 3—Setting up the Directory
Describes how to set up and maintain the Directory.

Chapter 4—Initiating the SUPRA Server Physical Data Manager
Describes how to enable the multitask SUPRA PDM, create an initiation script, use a prefix, create a PDM input file, define logical names, and use automatic startup.

Chapter 5—Communicating with the SUPRA Server PDM
Describes how to communicate with and write your own interface to the PDM.

Chapter 6—Tuning your database
Describes how to tune the database, design application programs, and control data item lists.

Chapter 7—Using the UNIX online Help facility
Describes how to enter the csihelp command, and how to create a Help file and Help index.

Chapter 8—Database migration
Describes how to migrate into and out of SUPRA Server, generate a DDL file, format data sets, and add records.

Chapter 9—System Administration utilities
Describes utilities used by the System Administrator to assist in tuning databases, PDM systems, and diagnose problems.

Appendix A—Example user exits
Presents example user exits written in COBOL and FORTRAN.

Appendix B—PDM statistics output
Describes the statistics written to the log file when you specify STATISTICS=Y in the PDM input file.

Appendix C—Example mailbox read program
Presents a C program to read the PDM messages from a named pipe.

Index

Revisions to this manual

The following changes have been included for this release:

- A bullet discussing automatic read-ahead buffering (see "The Physical Data Manager" on page 18)
- An additional semaphore set identifier (see the table under "Modifying system parameters for semaphores" on page 30):
  - Internal PDM lock of relative files with LOAD-LIMIT=0
- Information about multitask PDM (see "Installing SUPRA Server PDM" on page 34 and "Initiating the SUPRA Server Physical Data Manager" on page 81)
- Additional logical names (see the table under "Implementing logical names" on page 38):
  - CSI_BAK
  - CSI_BATCH_CONCURRENT
  - CSI_READAHEAD
  - CSI_READAHEAD_STATISTICS
- Two new parameters, READAHEAD_THRESHOLD1 and READAHEAD_THRESHOLD2, have been added to the PDM input file. See "Creating a PDM input file" on page 96.
- Information about the UNIX KILL command (see "Starting daemon processes" on page 53)
- Information about automatic PDM initiation (see "How to enable the multitask SUPRA PDM" on page 82)
- The BATCHTHREADS parameter (see "Creating a PDM input file" on page 96)
- Information about single-task PDM running in concurrent and stand-alone modes (see "Single-task PDM" on page 117)
- The REDO and QUIT commands have been added to the section on csiopcom commands on page 152

- A bullet discussing improving initial load performance (see “Managing buffers” on page 181)
- Information about improving database performance with buffers (see “Improving database performance with buffers” beginning on page 183)
- Information about tuning PDM process memory (see “Tuning PDM process memory” on page 186)
- Information about CONTROL:Manufacturing tuning considerations (see “CONTROL: Manufacturing tuning considerations” on page 187)
- Information about improving database performance with indices (see “Improving database performance with indices” on page 188)
- Information about record holding (see “Record holding” on page 190)
- Information about understanding read-ahead buffering (see “Understanding read-ahead buffering” on page 194)

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example

`Constant width type`	Represents screen images and segments of code.	`PUT 'customer.dat'`
		`GET 'miller\customer.dat'`
		`PUT '\DEV\RMT0'`

Slashed b (ⱥ)	Indicates a space (blank). The example indicates that four spaces appear between the keywords.	`BEGNⱥⱥⱥⱥSERIAL`
Brackets [ ]	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations:	
	A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a WHERE clause.	`[WHERE search-condition]`
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	**(WAIT)** **(NOWAIT)**
Braces { }	Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select. The example indicates that you must enter ON or OFF when using the MONITOR statement.	**MONITOR** { **ON** / **OFF** }

Convention	Description	Example
Underlining (In syntax)	Indicates the default value supplied when you omit a parameter. The example indicates that if you do not choose a parameter, the system defaults to WAIT.	[**(WAIT)** **(NOWAIT)**]
	Underlining also indicates an allowable abbreviation or the shortest truncation allowed. The example indicates that you can enter either STAT or STATISTICS.	STATISTICS
Ellipsis points...	Indicate that the preceding item can be repeated. The example indicates that you can enter multiple host variables and associated indicator variables.	INTO :*host-variable* [:*ind-variable*],...
UPPERCASE lowercase	In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.	COPY MY_DATA.SEQ HOLD_DATA.SEQ
	In the UNIX operating environment, keywords are case-sensitive, and you must enter them exactly as shown.	cp *.QAR /backup
Italics	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you must substitute the name of a table.	FROM *table-name*
Punctuation marks	Indicate required syntax that you must code exactly as presented. ( ) parentheses . period , comma : colon ' ' single quotation marks	(*user-id*, *password*, *db-name*) INFILE 'Cust.Memo' CONTROL LEN4
SMALL CAPS	Represent a keystroke. Multiple keystrokes are hyphenated.	ALT-TAB

SUPRA Server documentation series

SUPRA Server is the advanced relational database management system for high-volume, update-oriented production processing. A number of tools are available with SUPRA Server including DBA Functions, DBAID, precompilers, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed.

Getting started

- *SUPRA Server PDM UNIX Installation Guide*, P25-1008
- *SUPRA Server PDM UNIX Tutorial*, T25-2262

General use

- *SUPRA Server PDM Glossary*, P26-0675
- *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022

Database administration tasks

- *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260
- *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132*
- *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220
- *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- *SPECTRA Administrator's Guide*, P26-9220**

Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.

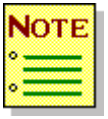
Application programming tasks

- *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240
- *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132*
- *MANTIS Planning Guide*, P25-1315**
- *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*

Report tasks

- *SPECTRA User's Guide*, P26-9561**

Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.



Educational material is available from your regional Cincom education department.

1

Overview of SUPRA Server

SUPRA Server is an interactive database system which allows you to use advanced features for control of data resources and high programming productivity. SUPRA Server accommodates the varying data processing needs of its users by supporting the following integrated components and related products:

Component/related product	Function
Physical Data Manager (PDM)	Recovery Automatic restart
Directory	DBA functions Fast utilities Database Verify utility Batch validate, compile and print Format data files Format, populate and check indices
MANTIS	Application Development

The Physical Data Manager

The Physical Data Manager (PDM) controls the storage of and access to data in user databases and is the underlying control method that all other components use to access physical data. Application programmers write programs using Physical Data Manipulation Language (PDML) to access the PDM and directly manipulate data held on SUPRA Server databases.

SUPRA PDM runs in multitask mode or single-task mode:

- The multitask SUPRA PDM runs as multiple detached processes. In this mode, application programs coded with PDML call the database access program (csidatbas.o or csidatbas.sl), which provides access to SUPRA PDM through shared memory or TCP/IP messages.

- The single-task SUPRA PDM runs in the application process. In this mode, application programs coded with PDML call the database access program (csibatbas.o), which provides high-speed access to the SUPRA PDM logic directly via a call (see "Installing SUPRA Server PDM" on page 34 and *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240).

Regardless of the access method, the PDM provides high performance through the following additional features:

- **Recovery methods**, including:

  - Task level recovery. Resets the database to the last successful commit point after a system or task failure.

  - System level recovery. Restores the database to its state just prior to a failure by reapplying logged after-images.

  - Shadow recording. Duplicates each data set write on a shadow data set, switches to duplicate data sets when a failure occurs.

  Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information about recovery methods.

- **Network support.** The PDM provides Network Support for each database, provided you define a preferred machine list. SUPRA Server Network Support enables a task from one machine to access a database running on another machine where both machines form part of the same network. “Initiating the SUPRA Server Physical Data Manager” on page 81 describes how to set up your PDM to start on any machine in a network in descending order of preference. This order of preference depends on the order of machines as specified in the preferred machine list.

- **Automatic restart.** Automatic restart ensures that your PDM will still run even if the machine on which it is currently running fails. The first task to attempt to access a database running in the failed PDM initiates the PDM in the first available machine on its preferred machine list. “Understanding automatic PDM startup” on page 111 describes automatic restart.

- **Automatic read-ahead buffering.** Automatic read-ahead buffering provides an invisible performance boost for sequential and serial reads executed against the same file. Read-ahead buffers are provided for the RDNXT, READV/READR, and READX PDML functions. This feature can provide dramatic performance improvements when applications read large numbers of records using the above functions.

The Directory

The SUPRA Directory is the central point for all SUPRA Server components. It supplies all the system information and all the data descriptions required by the PDM. Although it does not maintain or contain user physical data (the PDM fulfills this function), it provides a common repository of what data exists and where. “Setting up the Directory” on page 69 describes how to set up and modify the Directory. The DBA accesses the Directory to define and maintain database descriptions using these facilities:

- **DBA functions.** Database administrators use the DBA functions to define and maintain:

  - The Directory itself
  - Database descriptions stored on the Directory

  In addition, the Directory controls:

  - Database users
  - Access rights
  - Recovery functions
  - DBA utilities

  Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for details of how to use the DBA functions.

- **Fast utilities.** The DBA or System Administrator uses Fast utilities to make physical changes to databases, including the Directory database SUPRAD. You can use Fast utilities both online and in batch. Refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220, for details of both Fast utilities and DBA utilities.

- **Batch validate, compile, and print.** The batch validate, compile, and print program, called csmcombat, is supplied with the DBA validate, compile, and print functions. The DBA can invoke csmcombat from within SUPRA DBA or from the command level. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information.

- **Format data sets.** The format program, called cstufmt, runs from SUPRA DBA or from the command level. Format creates and formats the physical files that will hold the data. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information.

MANTIS

MANTIS is a programming language with facilities to design scenarios, files, screens, prompters, and interfaces. The MANTIS programmer can access all SUPRA Server components. MANTIS offers high programming productivity and fourth-generation prototyping facilities. In addition, MANTIS applications are portable across hardware platforms. Refer to MANTIS documentation for more information about using MANTIS.

2

Setting up UNIX to run SUPRA Server

Setting up UNIX to run SUPRA Server involves the following tasks:

- Modifying UNIX system parameters to accommodate very numerous or large databases (“Modifying UNIX system parameters” on page 24)
- Installing SUPRA Server PDM (“Installing SUPRA Server PDM” on page 34)
- Defining environment variables (“Defining environment variables” on page 37)
- Implementing the logical names needed to run SUPRA Server (“Implementing logical names” on page 38)
- Starting daemon processes (“Starting daemon processes” on page 53)
- Writing SUPRA Server PDM user exits (“Writing SUPRA Server PDM user exits” on page 66)

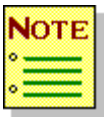
Modifying UNIX system parameters

To run SUPRA Server on your UNIX system you will need to check and possibly modify your system parameters. Since the procedures to modify UNIX system parameters can vary from system to system, consult your UNIX documentation set for the modification procedures.

You may need to modify these system parameters:

- Shared Memory
- Semaphores
- Files
- Message Queues

Use the byte requirements in this section to calculate the size you will need for the above parameters. The requirements given represent the bytes per database, PDM, or file system that you use.



UNIX system parameters are typically set at low values; therefore, do not be alarmed if you have to set the parameters at values that are significantly higher than those supplied with your system.

Modifying system parameters for shared memory

SUPRA Server relies on shared memory for communications, logical names, and shared data. In order to accommodate the shared memory used by SUPRA Server, the following shared memory parameters will probably require changing:

- **SHMMAX**. Maximum shared memory size
- **SHMMNI**. Maximum number of shared memory identifiers in the system
- **SHMSEG**. Maximum number of shared memory segment attaches per process

The names of the parameters above may vary among UNIX systems. Check your UNIX documentation for the correct parameter names for your particular system.

When the SUPRA PDM is started, it creates a shared memory segment containing all its constants plus its application communication areas. Each database loaded occupies two shared memory segments. Of the two created, one can be very large, depending on the size of the database in use (number of files, elements, etc.). In order to accommodate this size, it may be necessary to set maximum shared memory size (SHMMAX) to values in the order of megabytes. The dictionary database SUPRAD, for instance, creates a shared memory segment of 200K. If you plan on using a large number of databases, you may need to change the maximum number of shared memory segments allowed in the system (SHMMNI).

SUPRA Server has four different types of shared memory segments:

- Logical name table and directory table segments
- Global PDM data and message area segments
- Segments that the database occupies
- Tidy Daemon segment

The following information shows each type of memory segment and the requirements for each. Use these requirements to calculate the sizes you will need for the SHMMAX, SHMMNI, and SHMSEG parameters.

- **Logical name table and directory table segments.** The PDM creates a shared memory segment for each logical name table and directory table in the system. The number of logical name tables created is as follows:

Table	Number of tables created
System table	One per system
System directory table	One per system
Group table	One per group
Process table	One per parent process (process setting the CSIPID environment variable)
Process directory table	One per parent process (process setting the CSIPID environment variable)
User table	One per user-defined table created

Each logical name table requires the following memory:

40 byte header record + 336 bytes per logical name table entry (default is 50 entries)

The size of each table is rounded to the system page size. For example, if the system page size was 4096 bytes and a logical name table of 50 entries is created, its size would be 40 + (50 * 336). The resulting nearest page boundary is 16840; rounded to the nearest page is 20,480 bytes.

- **Global PDM data and message area segments.** The PDM creates a shared memory segment to contain global PDM data plus the message areas used to communicate with the applications. The size of this area can be calculated as follows:

  5396 bytes for global PDM data +

  (1344 + value of MAXDATA) page aligned *

  (MAXTASKS + 5) + (2 * MAXTASKS + 1) * 16

  The size of this area is then aligned to a page boundary; for example, if page size = 4096, MAXTASKS = 10 and MAXDATA = 8192, then the size of the area is:

  5396 + (1344 + 8192) page aligned * (15) + (21) * 16

  = 5396 + 184320 + 336

  = 190052 bytes

- **Segments that the database occupies.** SUPRA Server has two shared memory segments per database loaded. The sizes of these segments are shown in the following two tables:

Contents in segment 1	Number of bytes required
Header	1100
Database details	992
Data sets	292 * (*number-of-data-sets*)
Task log	284
System log	284
Elements (Base elements are duplicated for each record code)	48 * (*total-number-of-elements*)
Linkpaths	20 * (*total-number-of-linkpaths*)
Record code table	4 * (*total-number-of-record-codes*)
Record hold table	80 * (*Max-Held-Records* + *1*)
Tasklog Sector map header	20 (*used even if no Tasklog defined*)
Tasklog Sector map	24 * (*Max-Update-Tasks*)
Tasklog Sector group list	12 * (*Max-Update-Tasks*)
Task table	336 * (*Max-Tasks* + *1*)
Task schema	20 * (*number-of-data-sets*) * (*Max-Tasks*)
CCR buffers	20 * ((5 * *number-of-related-data-sets* or MAX-Tasks whichever is greater) + 32
Including extents for Task and System logs (double if shadowing)	112 * (*total-number-of-file extents*)
Buffers	32 * (*number-of-buffers*) + 511
Each buffer defined	104 + (*buffer-size* * *buffer-copies*)

Contents in segment 2	Number of bytes required
Function backout table	MAXTHREADS * 40 * (2*max-linkpaths-in-a-data-set* + 1)
Linkpath table	MAXTHREADS * 48 * (*max-linkpaths-in-a-data-set* + 8)
Thread context table	MAXTHREADS * 192
Record work areas	MAXTHREADS * 2 * (*max-record-size-in-database*)
Tasklog buffers	MAXTHREADS * 4 * (*Tasklog-block- size*) + PAGESIZE - 1
Tasklog buffer update table	MAXTHREADS * ((96 + *number-buffers* +(*number-related-data-sets* * 5) +*number-of-files* + 31) / 2 * 4)
File open tables (including Task and System Logs)	MAXTHREADS * 80 * (*number-of-data-sets*)
Thread process ID table	MAXTHREADS * 4

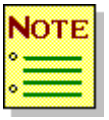
- **Tidy Daemon segment.** The tidy Daemon process creates a shared memory segment. The initial size of this segment will be 256 * 275 + 24 bytes (rounded to a page boundary).

Modifying system parameters for semaphores

SUPRA Server uses semaphores for the majority of its internal locking, synchronization, and task communications. In order to accommodate the semaphores SUPRA Server uses, you may have to change the following system parameters:

- **SEMMNI**. Number of semaphore identifiers
- **SEMMNS**. Maximum number of semaphores
- **SEMMSL**. Maximum semaphores per set
- **SEMMNU.** Maximum number of semaphore UNDO structures in the system.

SUPRA Server uses semaphore UNDO structures for most semaphore operations. The SEMMNU system parameter should be set to accommodate at least the total number of semaphores in all SUPRA Server systems. Use the table on the following page to determine the number of semaphores used by a SUPRA Server system. Remember, a SUPRA Server system can have many databases loaded.



The names of the parameters above may vary among UNIX systems. Check your UNIX documentation for the correct parameter names for your particular system. In addition, the number of maximum semaphores per set may be a fixed value.

A minimum of 12 semaphores will be generated. The actual number will expand depending on the number of tasks using the SUPRA PDM and the number of buffer pools defined in a given dbmod. Internally, SUPRA Server uses semaphore sets with a size of 25 (25 semaphores per set). This value may be modified by defining the logical name SEMMSL to be the set size desired (csideflog -g SEMMSL 50). SUPRA Server will create a file for each of the semaphore sets so it can obtain a unique semaphore ID. In order to accommodate this size, you may need to modify the values for the maximum number of semaphore identifiers (SEMMNI) and the maximum number of semaphores (SEMMNS).

The following table shows semaphore set identifiers that SUPRA Server uses. The total number of semaphores in use is the sum of the semaphore set identifier sizes.

Semaphore purpose	Set identifier size	Considerations
Internal PDM global locking	7	
Local task communication with PDM	MAXTASKS + 5	If MAXTASKS + 5 exceeds maximum semaphores per identifier, then multiple identifiers are used.
Internal PDM database locking	11 + number of buffer pools + 3	Per database loaded.
Internal PDM lock of related files with LOAD-LIMIT = 0	1	None.
Local PDM start	1	
Remote PDM start	1	Per machine where remote PDM is to run.
Logical Name locking	1	None.

Modifying system parameters for files

Depending on the size of the databases in use and the number of permitted threads in the PDM (MAXTHREADS), the PDM may have to open a large number of files simultaneously. Under these conditions, you may have to modify some of the file related system parameters to enable the PDM to work correctly.

Consider the following:

- The NFILE parameter should be at least (the number of files in each dbmod loaded in all PDM systems * the total number of MAXTHREADS for all PDM systems)+(3 * the number of PDM systems)
- The NINODE parameter should be equal to NFILE if possible
- The MAXFILES parameter should be at least (the maximum number of files in all dbmods loaded by a single PDM system + 3)

SUPRA Server uses file names that are longer than the default maximum in UNIX; therefore, it is necessary to turn on the long filenames option.

Modifying system parameters for message queues

SUPRA Server uses message queues for communications:

- To and from the operator and privilege checking daemon processes
- From the database access program (DATBAS) to the multitask PDM (csipdm)

You may need to change the following message queue system parameters:

- **MSGMAX**. Maximum message size in bytes
- **MSGMNI**. Maximum number of message queue identifiers
- **MSGMNB**. Maximum number of bytes for a message queue
- **MSGTQL.** Number of message headers in the system

MSGTQL must be set high enough to accommodate at least 1 message per task (MAXTASKS) for all SUPRA Server PDM systems.

The names of the parameters above may vary among UNIX systems. Check your UNIX documentation for the correct system parameter names.

For the privilege checking daemon (described in “Maintaining user privileges (csichkpriv)” on page 56), each request sent to and from it is 32 bytes long. Therefore, unless you run many SUPRA Server applications simultaneously, use the default system values. Only one message queue is created for the privilege daemon, with each privilege check sending one request on that queue.

Each PDM creates (if enabled) a message queue for the operator daemon to communicate with it. Entire PDM operator commands are sent to the PDM through the daemon. Therefore, it is unlikely any changes to the size parameters need to be made. However, if you run many PDMs and other applications that use message queues, you may need to adjust the maximum number of message queue identifiers.

Each multitask PDM creates two message queues. The database access program (DATBAS) uses the first queue (*systemname*_DAPMSG_QUEUE) to initiate functions in the multitask PDM (csipdm). The multitask PDM dispatch task uses the other queue (*systemname_dbmodname*_THDMSG_QUEUE) to initiate functions in the threads.

Installing SUPRA Server PDM

SUPRA Server PDM for UNIX and the utilities provided with it manage shared memory, semaphores, message queues, and file systems provided by the UNIX operating system.

SUPRA Server PDM defines the following directories during installation. For more information regarding installation, refer to the *SUPRA Server PDM UNIX Installation Guide,* P25-1008.

- **Install directory**. The directory into which the release tape is restored. Only one Install directory can be on a machine. The Install directory may contain any number of SUPRA PDM releases.

- **System directory**. A directory corresponding to a SUPRA PDM System. Each SUPRA PDM System has a 1- to 8-character name and may be either a systemwide or groupwide system. There may be any number of SUPRA PDM Systems on a machine. Each System directory contains the necessary files and scripts to manage a SUPRA PDM System.

- **Resource directory**. A directory used by a SUPRA PDM System to manage UNIX resources. Each shared memory segment, semaphore set, and message queue has a corresponding file in the Resource directory which maps to the resource by producing the unique key associated with the resource. One systemwide SUPRA PDM System and multiple groupwide SUPRA PDM Systems may share a single Resource directory. By defining multiple Resource directories, you can support multiple systemwide SUPRA PDM Systems.

- **Directory directory**. A directory which contains the SUPRA PDM Directory database. The Directory database contains your database definitions. The SUPRA PDM Database Administration Utility (csidba) maintains it. The SUPRA PDM Database Administration Utility compiles the database definitions defined in the SUPRA PDM Directory database to produce a database definition module (dbmod) for each database.

SUPRA PDM relies on UNIX environment variables and SUPRA PDM logical names to control the operating environment of a SUPRA PDM System. You control the execution characteristics of SUPRA PDM by defining values for the various environment variables and logical names used by SUPRA PDM.

The UNIX environment variable CSIRESOURCES defines the path to the Resource directory for a SUPRA PDM System. You must set the UNIX environment variable CSIPID to a valid process ID. This variable defines the process ID to be used for process-level logical name lookups. Each UNIX user who accesses a SUPRA PDM System must have these environment variables set.

You must define the SUPRA PDM logical name CSIPDMID as the name of the SUPRA PDM System. If this logical name is defined in a group table, the SUPRA PDM is groupwide. If the logical name is defined in a system table, the SUPRA PDM is systemwide.

A complete list of SUPRA PDM logical names appears later in this chapter.

The SUPRA PDM install scripts (s1_install, s1_system_setup, and s1_user_setup) simplify and automate the definition of SUPRA PDM Systems and the users who may access them. Refer to the *SUPRA Server PDM UNIX Installation Guide,* P25-1008, for more details on installing SUPRA PDM. The install scripts automatically create customized scripts (pdm_startup and daemon_startup) and input files for starting a SUPRA PDM System.

Each SUPRA PDM System requires several daemon processes to be executed. These daemon processes do the following:

- Manage access privileges
- Clean up UNIX system resources for failed application and PDM processes
- Establish communications with remote SUPRA PDM Systems on a TCP/IP network
- Provide a command-driven operator interface

The pdm_startup script starts these daemons automatically. Alternatively, the daemon_startup script is generated by the install scripts to initiate only the daemons.

The multitask PDM may be started using the pdm_startup script, or if the CSI_ AUTOSTART logical name is defined as YES, the first application to execute a SINON PDML function will start it automatically.

The multitask SUPRA PDM runs as several daemon processes. The parent process is called the Dispatcher. Its primary function is to monitor the database access program queue for Physical Data Management Language (PDML) requests produced by application program calls to the database access program (DATBAS). Then, a second SUPRA PDM process is created and is responsible for monitoring the application tasks which are currently using a SUPRA PDM System. If an application task fails without signing off of the database, this task signs off the application task off dynamically and rolls back any updated database records. The csipdm daemon creates these two SUPRA PDM processes automatically (see “Initiating the SUPRA Server Physical Data Manager” on page 81).

When the first application task signs on to a database, the Dispatcher task loads the database definition module (dbmod) into shared memory and creates a number of SUPRA PDM processes. The number of processes created depends on the value of the MAXTHREADS SUPRA PDM input parameter. The SUPRA PDM input file parameters are described in detail in this manual. Each dbmod loaded has this number of processes created.

Defining environment variables

You must define two environment variables for any process that needs to use a component of SUPRA Server. You can define the environment variables as part of the shell startup script, for example, the .profile for the source shell. You should export the environment variables so they are available to all child processes. The environment variables are as follows:

- **CSIPID** Equates to the process ID of the current process. This environment variable is used by the SUPRA Server logical name facilities. For example:

  ```
  CSIPID = $$; export CSIPID
  ```

  You must define the CSIPID environment variable in the profile or login file of each UNIX user who accesses a SUPRA PDM System. If you use scripts running batch applications which access SUPRA PDM, you must redefine this variable and the process logical names, such as CSI_PREFIX, in the script. Otherwise, when the user exits from the UNIX system (but the batch process stays active), the csitidy daemon removes the process logical name table from the system. Then, the batch application produces errors such as NMAC or “File not found” because SUPRA PDM cannot resolve the process logical names. See “Implementing logical names” on page 38 for more information.

- **CSIRESOURCES** Equates to the path in which all SUPRA Server resource files are to be created. Resource files are created by all SUPRA Server components plus your application. For example:

  ```
  CSIRESOURCES = /usr/acct/resource; export CSIRESOURCES
  ```

Implementing logical names

A logical name is a shorthand notation for representing a file specification, a value, or another logical name. Logical names allow you to keep programs and shell scripts independent of physical file locations, and to dynamically pass data to applications. The program or shell script will search and write to the logical name tables in a certain order unless you specify otherwise.

You can restrict logical name tables to use by a single process, or you can share them among many processes. The following logical name tables are available:

Table	Description
Process table	Contains logical names that are only available to your process and its children. When all processes with access to the table have been deleted, the table itself is deleted, for example, it only exists for the lifetime of the parent plus its children.
Group table	Contains logical names that are only available to processes with the same group (there is a separate group table for each group ID on the system). Any process outside of the group does not have access to the table unless it has root privileges. You must explicitly delete group logical name tables and their logical names.
System table	Contains logical names that are available to all processes on the system. You must explicitly delete the system logical name table plus its logical names.
User-defined tables	These tables are intended to support multiple systemwide PDMs. The user defines the availability of the tables. A child process will inherit a parent's user-defined tables plus access to them. You must explicitly delete user-defined logical name tables and their logical names.
Logical name directory tables	Catalog logical name tables and define their search order. Logical name directory tables are themselves logical name tables with catalogs in the form of logical names, for example, there is a logical name for each table in the system. There are two logical name directory tables named lnm_system_directory and lnm_process_directory which catalog the shareable and process- level logical name tables respectively.

The logical name search list is defined by the logical name lnm_file_dev. By default, this logical name is defined by the system directory table and is defined as lnm_process, lnm_group, lnm_system to indicate that the search order of the tables is process followed by group, and then by system. A process may define a local search list, for example, a search list unique to that process, by defining the lnm_file_dev logical name in its process directory.

The scope of the logical name is defined by the logical name table in which it appears. One logical name may appear in more than one logical name table, depending on how you set up your system. A logical name that occurs in more than one table may be associated with a different equivalence name in each logical name table.

The system searches logical name tables in a particular order and uses the first logical name found, regardless of any other occurrences.

The equivalence name represents a file or a value to be passed to an application. You normally make logical assignments through a shell script. For example, you could assign the logical name CSIPDM to the background PDM image at system level using the supplied csideflog command:

```
csideflog -s CSIPDM /home/bin/csipdm
```

Through a set of supplied utilities and object libraries, you can create, delete, examine, and modify both logical names and logical name tables from within programs and shell scripts. For information on how to create logical name tables, see “Creating logical name tables” on page 44.

The following table lists the logical names needed to run SUPRA Server. The Table column gives the tables in which you can place the logical name:

- **P**. PROCESS table
- **G**. GROUP table
- **P/G**. PROCESS or GROUP name table
- **G/S**. GROUP table if you are running a groupwide PDM; SYSTEM table if you are running a systemwide PDM
- **Any**. It does not matter which table, provided you define the logical name somewhere

Logical name	Table	Equivalence name	Function
CSIDAPLOG	G/S	*filename*	Log file for CSIDATBAS.
CSIDBAUTL	Any	SUPRA_EXE:csidbautl	Stand-alone DBA utilities program.
CSIHELP	Any	SUPRA_EXE:csihelp	Help utility.
CSIMVCORE	Any	*filename*	Name of script file executed during exception handling. Used to save core dump. The default script is csimvcore.
CSIOPCOM_AUTH	Any	*filename*	PDM command authorization file.
CSIPDM	G/S	SUPRA_EXE:csipdm	PDM image.
CSIPDMINP	G/S	*filename*	PDM input file.
CSIPDMLOG	G/S	*filename*	Log file for PDM messages.
CSISTRLOG	G/S	*filename*	Output file for startup program.
CSI_AUTOSTART	Any	YES or NO	Enable/disable automatic PDM autostart. Default = YES.
CSI_BAK	Any	YES or NO	Backup data set prior to executing csidba/expand or csm changedb. Default = YES.
CSI_BATCH_CONCURRENT	Any	YES	Causes single-task SUPRA PDM applications to run in concurrent mode (HP-UX and AIX platforms only).

Logical name	Table	Equivalence name	Function
CSI_CONSOLE	P/G/S	OPER*n*	Sends messages from csidap to the specified operator console.
CSI_DIRDB	G/S	path	Directory containing data dictionary.
CSI_DBA_PRINT	Any	DDL, Normal (both)	Defines the print style for DBA database listings.
CSI_DMPANL	Any	*filename*	Dump analysis file in case of crash.
CSI_HELPFILE	Any	*filename*	Help file for Help utility.
CSI_INHERIT_SINON	Any	YES or NO	Tells DATBAS to use a new TMS for child process. Default = NO.
CSI_INIT_*N*	S	*filename*	SUPRA Server initialization script for remote startups.
CSI_INTERVAL	Any	1-*n*	Interval for startup retries (both local and remote).
CSI_MRELAY	Any	TRUE or FALSE	Sends messages to named pipes.
CSI_*nnnnnn*	G/S	*filename*	Points to a PDM initiation script.
CSI_PDMID	G/S	1- to 8-character name	Name of the PDM.
CSI_PREFIX	Any	1- to 3-character prefix	Prefix used to distinguish databases of the same name running in the same group or system.
CSI_PRIVFILE	S	*filename*	SUPRA Server privilege file.
CSI_READAHEAD	Any	YES or NO	Use read-ahead buffering. Default = YES.
CSI_READAHEAD_ STATISTICS	Any	YES or NO	Print read-ahead statistics at sign off. Statistics are printed in CSIDAPLOG. Default = NO.
CSI_REPLYTIMER	S	1-*n*	Number of minutes between operator reply messages.
CSI_SERVICE	S	Must be the same for both client and server	Defines entry in /etc/ services file.

Logical name	Table	Equivalence name	Function
CSI_SYSPDMID	P	1- to 8-character name	Name of global section used by a systemwide PDM when the multiple, systemwide PDMs facility is used.
CSI_TIMEOUT	Any	1-*n*	Timeout for automatic startup (local and remote) in intervals.
CSI_START_RETRIES	Any	1-*n*	The number of times the PDM will attempt a startup before failing.
CSI_USEREX	Any	*filename*	PDM user exit.
CSMCOMBAT	Any	SUPRA_EXE:csmcombat	Batch compile.
CSTUDSLF	Any	SUPRA_EXE:cstudslf	System log dump program.
CSTUFMT	Any	SUPRA_EXE:cstufmt	Stand-alone format program.
CSTUIDX	Any	SUPRA_EXE:CSTUIDX	Path to index utilities program.
CSTURCV	Any	SUPRA_EXE:csturcv	Stand-alone recovery program.
*dbname*_CSI_PDM_MACS	Any	List of machines	Preferred machine list for PDM.
*dbname*_THREADS	Any	1-*n*	Specify alternate MAXTHREADS parameter for a given database.
DUMPSLF_*dbname*	Any	*filename*	File-containing system log input parameters.
SEMMSL	Any	1-*n*	Number of semaphores per set.
SUPRA_HELP	Any	/supra1/supra1.rel*x.x.x*/help	Help directory for SUPRA Server.
SUPRAD	Any	CSI_DIRDB:suprad.mod	Directory file.
SUPRA_SCRIPTS	Any	path	Directory containing DBA- support routines.
SUPRA_EXE	Any	path	Directory containing SUPRA Server images.

In addition to the logical names described in the preceding table, SUPRA PDM recognizes logical names in the FILE-SPEC field of the data set physical file definition in the Database Administration Utility (csidba). Use the following syntax:

```
logical-name:file-name
```

Example

```
csideflog -g PATH01 path
PATH01:file-name
```

In this example, the first line defines the logical name PATH01. The second line defines the FILE-SPEC field.

With this FILE-SPEC definition method, you can move or copy files to different directories without recompiling the database definition. Recompilation is not needed because you modify the logical name definition rather that the database definition.

Creating logical name tables

The system, group, and process logical name tables are already created for you. However, you must actually create a logical name table under two circumstances:

- If the default size of the system, group, and/or process tables is not large enough for your application. In this case, you must create a table with the appropriate size.
- If you want to create a unique, user-defined logical name table.

To create a logical name table, use the following syntax:

csicretab [-e *number-of-entries*] [-p *permissions*] [-s]

table-name

-e *number-of-entries*

Description *Optional*. Defines the number of logical name entries allowed in this table. The default value is 500 for sharable tables and 50 for nonsharable tables.

Format 1-*n* numeric characters preceded by -e

-p *permissions*

Description *Optional*. Defines the logical name table permissions. The default value is 0666 for sharable tables and 0600 for nonsharable tables.

Format Octal numeric value preceded by -p

-s

Description *Optional*. Specifies that the table is sharable.

table-name

Description *Required*. Identifies the logical name table to be created.

Example The following example defines a logical name table CSI_PDM_MYPDM that is sharable and has 700 entries:

```
csicretab -e 700 -s CSI_PDM_MYPDM
```

Defining logical names

You use the csideflog utility to define logical names. The format of this utility is:

csideflog [**−p** | **−g** | **−s**] **[-t *table-name*] *logical-name equivalence-name***

[**−p** | **−g** | **−s**]

Description *Optional*. Specifies the level of the logical name.

Options

-p Process (default)

-g Group

-s System

-t *table-name*

Description *Optional*. Specifies the table into which the logical name is to be placed.

Format 1–255 alphanumeric characters preceded by -t

logical-name

Description *Required*. Specifies the logical name to be created.

Format 1–255 alphanumeric characters

Consideration If you use neither the level nor the table option, place the logical name in the first table in the process' logical name table search list.

equivalence-name

Description *Required*. Specifies the name to which the logical name equates.

Format A valid UNIX file specification or another logical name

Examples

- In the following example, csideflog defines the logical name TESTDB to equate to the file testdb.mod in the process logical name table:

```
csideflog -p TESTDB  /usr/alec/test/testdb.mod
```

- In the following example, csideflog defines the logical name MARKDB in the logical name table mark_private:

```
csideflog -t mark_private MARKDB /usr/mark/data/markdb.mod
```

- In the following example, csideflog defines the logical name ALECDB in the first table in the logical name search list:

```
csideflog ALECDB  /usr/alec/databases/alec_database.mod
```

Displaying logical names

You use the csisholog utility to display logical names. The format of csisholog is:

csisholog [**-p** | **-g** | **-s**] **[-t *table-name*] [*logical-name*]**

[**-p** | **-g** | **-s**]

Description *Optional*. Specifies the logical name to be displayed.

Options

-p Process

-g Group

-s System

-t *table-name*

Description *Optional*. Specifies the table that you use to search for the logical name.

Format 1–255 alphanumeric characters preceded by -t

Consideration If neither the level or table options are used, all levels and all tables are searched in the order of the process search list.

logical-name

Description *Optional*. Specifies the logical name to be displayed.

Format 1–255 alphanumeric characters

Considerations

- If no logical name is specified, all logical names in the table or level are displayed.
- Wildcard characters may be used to search for a group of logical names with a certain pattern of characters.

Examples

- The following command displays all occurrences of the logical name steph from the process' current logical name table search list:

```
csisholog steph
```

- The following command displays all logical names in the process logical name table beginning with the letter *a*:

```
csisholog -p a*
```

Displaying logical name values

You use the csiecolog utility to display the value of a single logical name. csiecolog is particularly useful in using logical names in a script. The format of csiecolog is:

csiecolog [**-p** | **-g** | **-s**] **[-t *table-name*] *logical-name***

[**-p** | **-g** | **-s**]

Description *Optional*. Specifies the logical name to be displayed.

Options

-p Process

-g Group

-s System

-t *table-name*

Description *Optional*. Specifies the table that you use to search for the logical name.

Format 1–255 alphanumeric characters preceded by -t

Consideration If neither the level nor table options are used, all levels and all tables are searched in the order of the process' search list.

logical-name

Description *Required*. Specifies the logical name to be displayed.

Format 1–255 alphanumeric characters

Considerations

- If the logical name specified is not defined, nothing is displayed.
- Wildcard characters may be used to search for a group of logical names with a certain pattern of characters.

Examples

- The following command displays the first occurrence of the logical name CSIPDM from the process' current logical name table search list:

```
csiecolog CSIPDM
```

- The following command loads an environment variable with the contents of a logical name:

```
CSIPDM_PATH = `csiecolog CSIPDM`
```

Deleting logical names

You enter the csidellog utility to delete logical names. The format of the csidellog utility is:

csidellog [**−p** | **−g** | **−s**] **[-t *table-name*] *logical-name***

[**−p** | **−g** | **−s**]

Description *Optional*. Specifies the logical name to be deleted.

Options -p Process

-g Group

-s System

-t *table-name*

Description *Optional*. Specifies the table from which the logical name is to be deleted.

Format 1–255 alphanumeric characters preceded by -t

logical-name

Description *Required*. Specifies the logical name to be deleted.

Format 1–255 alphanumeric characters

Considerations

- No pattern matching is supported. For example, you must specify the full logical name.
- If you use neither the level nor the table options, delete the logical name from the first table in the process' logical name table search list. Logical names are always displayed in alphabetic order.

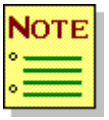
Modifying logical names

You can use the csideflog utility to modify logical names. When using csideflog, you specify the existing logical name in the logical-name parameter, and you modify the other parameters as you require. The csideflog utility then replaces the old parameters for the logical name with the ones you just specified. See “Defining logical names” on page 45 for the format of the csideflog utility.

Starting daemon processes

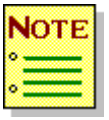
SUPRA Server supplies four daemon processes as described in the following table :

Daemon process	Purpose/ section	What to do before starting the daemon	Quantity	Required/ optional
csitidy	Removes resources used by processes that aborted or exited (see "Removing wasted resources (csitidy)" on page 55).		One per Resource directory	Required
csichkpriv	Maintains user privileges (see "Maintaining user privileges (csichkpriv)" on page 56).	1. Use the csisetpriv program to create a privilege file in which you define privileges for each user on the system. 2. Create a logical name in the system logical name table that points to the privilege file.	One per Resource directory	Required
csistr	Server for remote TCP/IP Client-Server operation (see "Connecting to a remote PDM—client/server (csistr)" on page 63).		One per system	Optional
csioper	Enables communication between processes and nominated operators (see "Enabling communication between processes and nominated operators (csioper)" on page 65).	Create a logical name in the system logical name table.	One per Resource directory	Optional



Start these daemons at system startup to prevent any problems that may occur with users who attempt to use SUPRA Server components without these processes running. Execute daemon_startup in the UNIX rc script, which executes during UNIX startup.

To start the daemon processes, execute either the pdm_startup or daemon_startup script. The install scripts generate those scripts when you define the SUPRA PDM System. For more information on the install scripts, refer to the *SUPRA Server PDM UNIX Installation Guide,* P25-1008.



The UNIX user who starts the daemon processes must have adequate permissions to perform the system operations the daemon processes will perform. To have adequate permissions, the root user must start the daemons.

The daemon processes also write information to log files. Each daemon has its own logfile in the $CSIRESOURCES directory. The name of the file is the name of the daemon plus the log name.

The UNIX KILL command is used to stop the daemons. Use the TERM signal (KILL -TERM *process-number*) for the best results on all platforms. The csiremall script also stops the daemons.

Removing wasted resources (csitidy)

The csitidy daemon is the SUPRA Server background resource tidy daemon. Its responsibility is to remove any resources that have been obtained by components of SUPRA Server or applications using SUPRA Server that have aborted or exited without removing these resources. Each component of SUPRA Server communicates with the tidy daemon through a named pipe, informing it of any resources that should be deleted when the program aborts or exits. The information regarding these resources is held in shared memory by the tidy daemon. Therefore, if the tidy daemon aborts abnormally for some reason, it can be restarted without loss of information, for example, it will use the information held in the shared memory segment used by the abnormally terminated daemon.

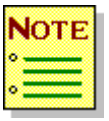
The tidy daemon accepts one optional parameter. This parameter defines the number of entries to allow for space in the shared memory segment. The default value is 1024; this value is normally adequate. If your pdm system has a large number of resources, you may want to increase this number; if your pdm system is smaller, you may want to decrease it.

The tidy daemon writes error messages to the csitidy.log log file in the $CSIRESOURCES directory. One csitidy daemon is required for each resource directory.

Maintaining user privileges (csichkpriv)

The csichkpriv daemon is a privilege-checking daemon that allows a multilevel security system to be applied to facilities within SUPRA Server. Standard UNIX only supports two levels of privilege: a normal user and the superuser. SUPRA Server has many facilities that should not be made available to a normal user. However, these facilities should not be restricted to a superuser, since any damage that could be caused through misuse is limited to SUPRA Server and not to other non-SUPRA Server users on the system. One privilege daemon is required for each resource directory.

The privilege daemon is a process that: (1) accepts privilege checking requests from applications, and (2) validates whether a particular user can perform a function by checking a privilege file you create with the csisetpriv program.



Anyone logged in as superuser has all privileges and effectively bypasses all checking.

An example of a function that requires a privilege within SUPRA Server is creating a system level logical name. Because logical names in the system logical name table are available to any user on the system, you would not want any user deleting and creating logical names in this table. You may wish to restrict this to your SUPRA DBA.

Understanding user privileges

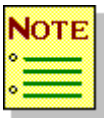
The following table lists the user privileges and provides a description of each. If a user attempts to use an unauthorized function, the function fails and the user receives a privilege violation.

Privilege	Description
DBAPRV	Allows the user to run csioauth, the csiopcom authorization program.
GRPCLN	Allows the user to remove any group resources that are not needed.
GRPLOG	Allows the user to create logical names in his/her group logical name table.
REPLY	Allows a user to use csireply operator communication facility. It is possible to talk to any SUPRA Server PDM using this facility. However, since this facility can destructively affect other users, it has been deemed a privileged function.
SETPRV	Allows the user to set privileges.
SYSCLN	Allows the user to remove any system resources that are not needed.
SYSLOG	Allows the user to create logical names and system directory logical name tables in the system.

The privilege-checking daemon accesses the user-privilege file through the logical name CSI_PRIVFILE, which should equate to the full path and file name of the privilege file.

Creating and modifying the privilege file

You use the csisetpriv program to create and modify a user privilege file.



To run this program, you must be logged in as superuser.

To run the csisetpriv program, enter csisetpriv. The program displays an introduction banner. If the logical name of the privilege file CSI_PRIVFILE does not exist, you will be prompted for the path of the privilege file. If this file (from logical or prompt) does not exist, you will be asked if you wish to create it. The csisetpriv program will show it is ready to accept commands by the "CSISETPRIV>" prompt. At this prompt, the following commands are available:

Commands	Description
ADD	Add a new user to the privilege file.
DISPLAY	Display details of a user(s) in the privilege file.
EXIT	Exit from the csisetpriv program.
HELP	Invoke the online Help facility. (After the word *help*, you can enter the name of the command you want help with, or just type help and you will get a Help screen.)
MODIFY	Modify an existing user in the privilege.
QUIT	Exit from the csisetpriv program.
REMOVE	Remove an existing user from the privilege file.

A description of each command and its format follows.

Adding a new user (ADD)

Use the ADD command to add a new user to the privilege file with the specified privileges. Enter the command in this format:

ADD USER=*user privilege-list*

user

Description *Required*. Specifies the name of the user to be added to the privilege file.

Format Valid user name on your system

privilege-list

Description *Required*. Specifies the list of privileges given to the user.

Considerations

- Separate each privilege with a blank space.
- You can use the keyword ALL to specify all privileges.
- See the table under “Understanding user privileges” on page 57 for a list of privileges.

Examples

- The following example gives the user *mark* the privilege to create group- and system-level logical names:

```
CSISETPRIV> ADD USER=mark SYSLOG GRPLOG
```

- The following gives the user *steph* all privileges:

```
CSISETPRIV> ADD USER=steph ALL
```

Modifying privileges (MODIFY)

Use the MODIFY command to modify the privileges of an existing user in the privilege file. The command takes the following format:

MODIFY USER=*user privilege-list*

user

Description *Required*. Specifies the name of the user to be modified in the privilege file.

Consideration Valid user name on your system.

privilege-list

Description *Required*. Specifies the list of privileges to be added to or removed from the user.

Considerations

- If a list of privileges is given with repeating privileges, the last occurrence will be used. Therefore, if you add a particular privilege, then remove it in the same command, it will be removed.
- Use the keyword ALL to specify all privileges.
- To remove a privilege, precede its name with NO; for example, NOSYSLOG, NOPRVIO, NOALL, and so on, separating each privilege by a blank space.
- See the table under “Starting daemon processes” on page 53 for a list of privileges.

Examples

- The following command adds DBA privilege to the user *mark* but removes system logical name privilege:

```
CSISETPRIV> MODIFY USER=mark DBAPRV  NOSYSLOG
```

- The following example gives the user *steph* all privileges:

```
CSISETPRIV> MODIFY USER=steph ALL
```

Removing a user from the privilege file (REMOVE)

Use the REMOVE command to remove a user from the privilege file. The user is completely removed and is left without any privileges. The format of the command is as follows :

REMOVE USER=*user*

user

Description *Required*. Specifies the name of the user to be removed from the privilege file.

Consideration Valid user name on your system.

Example The following example removes the user *mark* from the privilege file:

```
CSISETPRIV> REMOVE USER=mark
```

Displaying user details within the privilege file (DISPLAY)

The DISPLAY command displays privileges of users within the privilege file. The format of the command is:

DISPLAY USER=*user*

user

Description *Required*. Specifies the name of the user to be displayed from the privilege file.

Considerations

- You can use the wildcard character (*) to display details of all users.
- If you do not use the wildcard character, use a valid user name on your system.

Examples

- The following example displays privilege details of the user *mark*:

```
CSISETPRIV> DISPLAY USER=mark
```

- The following example displays details of all users in the privilege file:

```
CSISETPRIV> DISPLAY USER=*
```

Connecting to a remote PDM—client/server (csistr)

The client/server feature of SUPRA Server PDM provides applications with TCP/IP communications with server PDMs. The standard DATBAS csidatbas.o or csidatbas.sl provides the client software. The csistr daemon provides the server software. The daemon runs on the server machine and provides access to any PDM running on that machine. If you want to start a PDM with remote access capabilities on a machine on your network, you must also run the csistr daemon on it.

The root user executes the csistr daemon on the server. It also runs any number of PDMs, the csitidy and csichkpriv daemons, and optionally the csioper daemon. The UNIX environment variable CSIRESOURCES must be defined on the server. The client UNIX user ID must be duplicated on the server. The client machine runs its own csitidy and csichkpriv daemons. The client must specify the database logical names and the server's PDM system name in the logical name CSI_PDMID. The database and its data sets reside on the server. Following are examples of logical name definitions on the client:

```
csideflog -g CSI_PDMID pdm-system-name
csideflog -g dbname /path/dbname.mod
csideflog -g dbname_CSI_PDM_MACS host-name
```

If databases are prefixed, the logical name CSI_PREFIX must be defined on both the server and the client. The client/server feature uses TCP/IP sockets, which require the definition of the supra1 service in /etc/services. The default value for the supra1 service is 8000. If this value conflicts with any existing service on either the client or the server, you may override the default value by specifying a different value for the supra1 service in /etc/services on the client and the server. For example:

```
supra1 8000/tcp      as defined in /etc/services
```

If you must run more than one csistr daemon, you can override the supra1 service name by using the CSI_SERVICE logical name. For example:

```
service-name 10000/tcp      as defined in /etc/services
csideflog -g CSI_SERVICE service-name   logical name definition
```

Service-name and its corresponding number must match on the client and the server.

You start the csistr daemon on the server at the same time as the other daemons. Because csistr refers to several logical names, execute it *after* defining the logical names. The daemon_startup and pdm_startup scripts can execute csistr.

There is no need to link applications with a special datbas. The preferred machine list logical name (*dbname*_CSI_PDM_MACS) specifies the host name for the location of the PDM, either local or remote. The /etc/hosts file or the name server defines the host name. When an application tries to sign-on to a database whose preferred machine list says that it should run on a remote node, the application attempts to communicate to the csistr daemon on that machine through the supral service TCP/IP socket. When it makes the connection, the csistr daemon generates a child process which opens a socket. This process performs database access functions for the client by calling the standard database access program, csidatbas, and communicates with the client using the socket created. If the PDM is not running on that remote node, automatic PDM startup occurs (as described in “Understanding automatic PDM startup” on page 111). When the client application issues a SINOF, the process terminates.

Enabling communication between processes and nominated operators (csioper)

The csioper daemon is a communications daemon that allows communications between processes and nominated operators. On each UNIX system, only one csioper process should be running. Operators are numbered 1–12 and prefixed by “OPER”, for example, OPER1, OPER2...OPER12, which you can assign to any number of terminals on the system. You use the csireply command to assign operator numbers. See “Communicating with the PDM using the csireply command” on page 160 for the procedures for using the csireply command.

Processes running on the system can send messages to assigned operator terminals and can also send requests that request a reply. Following are some operator terminal assignments:

```
OPER1 assigned to tty00 and ttypa
OPER9 assigned to tty01
```

In the above example, any messages that were sent to OPER1 would appear on terminals tty00 and ttypa, and any messages sent to OPER9 would appear on terminal tty01. You can also assign more than one operator number to one terminal.

Writing SUPRA Server PDM user exits

User exits are invoked in two different ways. Internal user exits are "called" in the DATBAS module. You can code two pre-PDM, internal user exit modules and a post-PDM, internal user exit module. External user exits are "fork and exec'ed" in the CSI_DATBAS module. You can code a pre-PDM external user exit module and a post-PDM, external user exit module.

Internal user exits

To implement internal user exits for UNIX PDM, perform the following steps:

1. Using the skeleton internal user exit program provided on the release tape in the file release/src/csiintuser.c, add code to the modules provided. If you write your modules in COBOL, simply add a function call to your entry points in csiintuser.c. csiintuser.c contains three entry points detailed as follows:

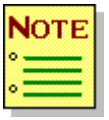
	Entry point name	Point of initiation	Return value
1	CSI_INT_USEREX1	Before DML Processing	None
2	DATBASXT	Before DML Processing	0, 4 or 8
3	CSI_INT_USEREX2	After DML Processing	None

All of the entry points are passed nine parameters which are addresses of the DML function parameters.

The DATBASXT exit can be used to modify a command before the DML is processed. From the DATBASXT module, it is possible to call CSI_DATBAS one or more times. This can be useful for implementing user-defined DML such as DELVC. A DELVC function would generate a READV and a DELVD for each record in a chain. You may code a return value for DATBASXT, which controls the processing, as follows:

Return code	Processing
0	Process the function normally; return to the application after calling CSI_INT_USEREX2.
4	Process the function normally; return to the DATBASXT module.
8	Do not process the function; return to the application after calling CSI_INIT_USEREX2.

2. Compile the source module csiintuser.c and link it with your application modules and CSIDATBAS to form an executable image.



If you want to use only one of the exits, the others must still be defined. They will simply return immediately.

External user exits

To implement external user exits for UNIX PDM, perform the following steps:

1. Using the skeleton, external user exit program provided on the release tape in the file release/src/csiextuser.c, add code to the modules provided. If you write your modules in COBOL, simply add function calls to your entry points in csiextuser.c. csiextuser.c contains two entry points defined as follows:

	Entry point name	Point of initiation	Return value
1	CSI_INT_USEREX1	Before DML Processing	None
2	CSI_INT_USEREX2	After DML Processing	None

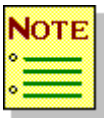
   All of the entry points are passed nine parameters which are addresses of the DML function parameters.

2. Compile the source module csiextuser.c and any other source modules that you have written. Link these modules with the object module release/bin/csiextmain.o to form an executable image.

3. Define the logical name CSI_USEREX to point to the executable image created above as follows:

```
csldeflog  -g  CSI_USEREX  /users/mark/mark.exit
```

   This logical name can be defined in any logical name table. If you define it in a shareable table, any application may use it.



If you want to use only one of the exits, the others must be defined. They will simply return immediately.

3

Setting up the Directory

The SUPRA Directory is the central source of control for your database. Setup and maintenance of the Directory involves the following:

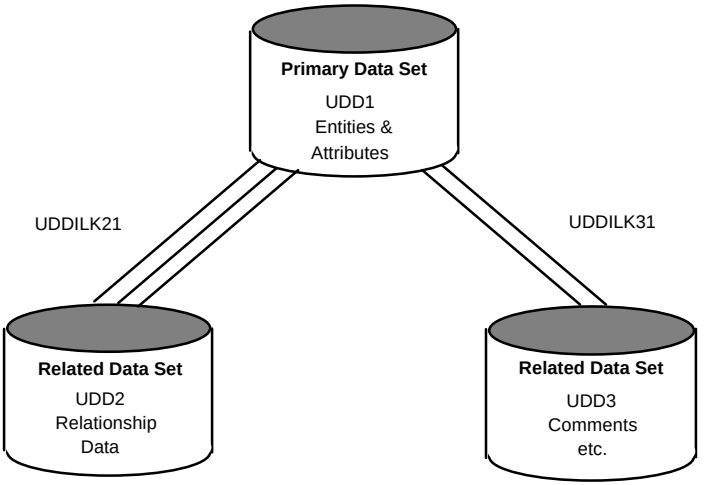
- Describing the data sets and linkpaths that comprise the SUPRAD database, space requirements, and supplied user names
- Changing the definition of the Directory database by listing permissible changes and explaining how to make them
- Modifying the Directory by changing the database description
- Modifying the Directory data sets by increasing the size or changing the load limit

Describing the Directory database

The SUPRA Directory is contained in the database SUPRAD. SUPRAD consists of the following data sets plus a task log:

- **UDD1**. A primary data set with entities and some attributes
- **UDD2**. A related data set with relationships between entities
- **UDD3**. A related data set with comment information

The following figure shows the structure of the SUPRAD database in more detail. The lines in the figure represent linkpaths between the data sets. Specify the primary linkpaths, UDD1LK21 and UDD1LK31, when running UNLOAD/RELOAD or the statistics utility from the Utilities function. The SUPRAD password is IONARY.



Because the Directory contains a substantial amount of information when it is delivered, ensure that SUPRAD is large enough for your initial needs. The next section, "Estimating Directory data set sizes," gives guidelines on Directory space requirements.

Estimating Directory data set sizes

The Directory data sets, like your private data sets, operate more efficiently if they are not too full. Use the following general guidelines to estimate the sizes of the Directory data sets for your system:

- Each entity requires two records on UDD1, one record on UDD2, and one record on UDD3.
- Each relationship requires one record on UDD2. Each entity has at least one relationship, and many have more. You can estimate the number of relationships as the number of entities times 4.
- UDD2, therefore, requires two and a half times as many records as UDD1.
- Each line of comment or navigation definition requires one record on UDD3.
- The Directory contains special information used by Directory maintenance and the definition of SUPRAD. Allow for this in your estimates as follows:

Directory data set	Directory maintenance information	SUPRAD definition	Total records required by data set	Total records available at install	UNIX bytes required
UDD1	500	200	700	5000	600K
UDD2	1100	450	1550	10,020	156K
UDD3	2500	300	2800	8000	1024K

Setting up Directory user names

The Directory contains two initial user names, both with blank passwords:

- **DATABASE-DESCRIPTIONS**. Can access all the databases, views, and so on, you will define
- **DATA-DICTIONARY**. Can access only the definition of the Directory, SUPRAD

Sign on to DBA with the user name DATABASE-DESCRIPTIONS to create other, less-privileged user names for your staff. Then change the passwords of the two initial user names. They do not have passwords defined in the standard directory.

To execute FORMAT, DBA Utilities, or RECOVERY, you need to sign on with the user name UTILITIES. The user name UTILITIES is not stored on the Directory. Use UTILITIES only when running the FORMAT, DBA Utilities, or RECOVERY functions against the Directory. The UTILITIES password is DATARULE. Because this user name is not stored on the Directory, you cannot change the password.

Changing the definition of the Directory database

You can change certain parts of the definition of the Directory database, SUPRAD. This database is defined on the Directory, and you must sign on to DBA with the user name DATA-DICTIONARY to modify it. You cannot alter the structure of SUPRAD, but you can change the following characteristics:

- Data set size(s)
- File specifications
- Shadow option
- System logging
- Number of buffers
- Maximum number of update tasks or processes allowed

You may wish to change specific Directory details, such as buffer allocation or logging options, at installation. Alternatively, after you have been running the SUPRA Directory for some time, you may need to modify the Directory definition if:

- A LOAD status occurs when you sign off from DBA. This indicates that one or both of UDD2 and UDD3 is at load limit.
- A FULL status occurs when you add or save a new database, a new data item, or a new view. This indicates that at least one of the UDD files has no free space left.

To check on the status of the UDD files, sign on to DBA with the user name DATA-DICTIONARY and run the statistics utility. Refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220, for details of how to produce statistics.

Modifying the Directory

The Directory is central to the operation of SUPRA Server. Follow these steps before modifying your Directory:

1. Ensure that no user is signed on to the Directory.

2. Make a backup copy of the Directory database and the SUPRAD compiled database description using the tar or cpio commands.

Keep a copy of a listing of the Directory database to be sure of its original details.

To modify the Directory database (SUPRAD), take these steps:

1. Sign on as user name DATA-DICTIONARY.

2. Make your changes, referring to the following table for required actions.

3. Validate and compile SUPRAD.

   Compiling SUPRAD generates a new SUPRAD.MOD; however, SUPRA Server continues to use the previous SUPRAD.MOD until all users have signed off. When a user signs on again either implicitly (after using FORMAT) or explicitly, SUPRA Server uses the new SUPRAD.MOD.

4. Take a backup of the new Directory and its compiled database description.

If you have a system log for SUPRAD.MOD, you must format a new system log after making your Directory modifications. This is essential if you change the system log size, any allocations, or data set details. The following table lists all possible changes to SUPRAD and the corresponding actions required after compilation.

SUPRA DBA uses the Directory to operate. During maintenance, the Directory might not be valid. Therefore, to sign on to SUPRA DBA without signing on to the Directory, use the special user name UTILITIES. The user name UTILITIES gives access only to FORMAT, UTILITIES, and RECOVERY. Use these functions to re-create a valid Directory, then sign on as usual.

If you select FORMAT, UTILITIES, or RECOVERY from a user name other than UTILITIES, the system signs off for you, even if you do not actually run the utilities. If you select another function, SUPRA Server signs on again using your original user name. Therefore, when using FORMAT, UTILITIES, or RECOVERY to maintain the Directory, always do so with the user name UTILITIES.

The following table lists the database details on the Directory that you can change and the actions you must take if you make the change:

Parameter type	Parameter name	Action required as a result of the modification
Database Details	DATABASE-PASSWORD	No further action.
	MAX-HELD-RECORDS	Do not reduce this value from 200.
	MAX-TASKS	Format a new task log. The size of the task log may change. The value it holds must match that in the compiled database description.
	MAX-UPDATE-TASKS	Format a new task log.
	SHADOW-OPTION	If you changed this from N, make a copy of the relevant data sets, using the names used in the file specifications.
	SINGLE-TASK	If you change this to Y, SUPRA Server sets MAX-TASKS and MAX-UPDATE-TASKS to one. Therefore, format a new task log. If you changed it to N, no further action is required unless you change MAX-TASKS or MAX-UPDATE-TASKS.
Task Log Details	TASK-LOG-BLOCK-SIZE	Format a new task log.
	TASK-LOG-NO-OF-BLOCKS	Format a new task log.
	TASK-LOG-NO-OF-BUFFERS	No further action.

Parameter type	Parameter name	Action required as a result of the modification
Task Log Details (*cont*.)	TASK-LOG-FILE-SPEC	Copy or rename the existing task log to the specified file, or format a new task log.
	TASK-LOG-SHADOW-FILE-SPEC	Copy the existing task log or format a new one.
System Log Details	SYSTEM-LOG-BLOCK-SIZE	Format a new system log.
	SYSTEM-LOG-NO-OF-BLOCKS	Format a new system log.
	FILE-1-FILE-SPEC	If just created, format a new system log.
	FILE-2-FILE-SPEC	If changed, copy the existing system log or format a new one.
	FILE-1-SHADOW-FILE-SPEC	Copy the system log or format a new one.
	FILE-2-SHADOW-FILE-SPEC	Copy the system log or format a new one.
Data Set Buffers	NUMBER-OF-COPIES-OF-BUFFER	No further action.
	BUFFER USE (which data sets share which buffers, including deleting buffers and creating new buffers).	No further action.
Data Set Details	TOTAL-LOGICAL-RECORDS	Unload and reload data set.
	LOGICAL-RECORDS-PER-BLOCK	Unload and reload data set.
	CONTROL-INTERVAL	Unload and reload data set.
	LOAD-LIMIT	Unload and reload data set.
	RECORD CODES	Cannot be changed.
	DATA ITEMS	Cannot be changed.
Data Set File Specifications	ALLOCATION - 1/2/3/4	Unload and reload data set.
	FILE-SPEC - 1/2/3/4	Copy or rename data set.
	SHADOW-FILE-SPEC - 1/2/3/4	Copy or rename data set.

Modifying the Directory data sets

You can increase the size or change the load limit of Directory data sets in UDD1, UDD2, and UDD3 in the following ways:

- Change the TOTAL-RECORDS and LOAD-LIMIT qualifiers of Fast utilities (see “Using Fast utilities on UDD files” on page 78)
- Use the unload and reload function from DBA utilities (see “Using DBA utilities on UDD files” on page 80)

You cannot use the Expand and Reset functions to modify Directory data sets UDD1, UDD2, and UDD3 because Expand and Reset update both the Directory and the data sets.

Do not confuse the old SUPRAD.MOD and new SUPRAD.MOD files. If UDD1, UDD2, and UDD3 do not exactly match SUPRAD.MOD, do not sign on to DBA, explicitly or implicitly, except with the user name UTILITIES.

Using Fast utilities on UDD files

Before you execute the Fast utilities against UDD files, take a backup of your Directory. To change the Directory data sets using Fast utilities, enter the command changedb at the shell command line together with a list of qualifiers. You must specify the user name DATA-DICTIONARY to modify the Directory. For example, to increase the file size and reset the load limit for the related data set UDD2, enter the changedb command in the following format:

```
csmchangedb
DATASET=UDD2
RELATED TOTAL_RECORDS=20000
LOAD_LIMIT=90
DB_NAME=SUPRAD
USERNAME=DATA-DICTIONARY
PASSWORD=IONARY
```

Alternatively, to alter more than one Directory data set at a time, use an ASCII line-terminated file containing a list of modifications. For example, the following change file modifies UDD1, UDD2, and UDD3:

```
DATASET=UDD1 PRIMARY TOTAL_RECORDS=10000
DATASET=UDD2 RELATED TOTAL_RECORDS=20000 LOAD_LIMIT=90
DATASET=UDD3 RELATED TOTAL_RECORDS=10000
DB_NAME=SUPRAD
USERNAME=DATA-DICTIONARY
```

You may include only data-set parameters in a change file. You cannot include the Directory Access parameters USERNAME and PASSWORD, or the database parameters SIGNON_DB_NAME, DB_PASSWORD, and OUTPUT. Specify these parameters on the command line or in a shell script.

You can also alter multiple Directory data sets using a shell script. For more information on doing this, refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220.

Invoke Fast utilities with a change file as follows:

```
csmchangedb  CHANGE_LIST= /path/uddchange.dat DB_NAME=suprad
```

If you wish to use more than one physical file for any UDD data set, remember that each UDD data set has a default file allocation of (1,0,0,0). For example, all the records held in the first file specified and none in the other three. You will have to modify the file allocation value to reflect the number of physical files you define for each UDD data set. Fast utility usage, including details of file allocations, are described in more detail in the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220.

Always make a backup of the Directory before you attempt any modifications.

Using DBA utilities on UDD files

To unload/reload data sets using DBA utilities, perform the following steps:

1. Copy the existing SUPRAD.MOD file to another physical file (cp suprad.mod oldsuprad.mod).

2. Sign on to DBA using user name DATA-DICTIONARY and apply the required changes. Change only one data set at a time.

3. Validate and compile the database SUPRAD. This will create a new SUPRAD.MOD file.

4. Exit from DBA and sign on again with user name UTILITIES.

5. Select the Utilities function from the main menu.

6. Select the unload/reload function from the utilities menu (function 1 for primary data sets or function 2 for related data sets). Specify OLDSUPRAD.MOD as the source database file specification, and the new SUPRAD.MOD (created in Step 3) as the target.

7. Repeat steps 2–6 for each data set you want to change. You must invoke several utility jobs, but you need to modify, validate, and compile the SUPRAD database only once for the primary data set UDD1, and once for the related data sets UDD2 and UDD3.

To unload and reload more than one Directory data set, first ensure that only one utility job at a time runs against the Directory. When the utility jobs finish, you can use the updated Directory. Refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220, for a detailed description of the function of DBA utilities.

4

Initiating the SUPRA Server Physical Data Manager

The multitask SUPRA Server Physical Data Manager (PDM) runs as several UNIX daemon (background) processes. The application tasks and the PDM processes communicate through the database access program, csidatbas, which is linked as part of the application program. The application programmer codes Physical Data Management Language (PDML) calls to the database access program's entry point, DATBAS. The database access program transfers these calls to the SUPRA PDM processes through:

- Shared memory, if the SUPRA PDM system is local
- TCP/IP messages, if the SUPRA PDM system is remote

When processing is complete, the SUPRA PDM processes return the results of the PDML functions to the database access program (DATBAS). Then, the database access program returns the results to the application program.

You initiate the multitask SUPRA PDM either manually or automatically as described later in this chapter. By entering operator commands through the Operator Communications Utility (csiopcom), you deinitialize the multitask SUPRA PDM.

The SUPRA PDM writes error messages to the standard error file (stderr) and a log file defined by the logical name CSIPDMLOG. By default, the standard error file is redirected to system-name.err in the Resource directory. The PDM log file is named system-name.log in the Resource directory.

The database access programs (csidatbas and csibatbas) write messages to the DAP log file defined by the logical name CSIDAPLOG. By default, this file is *system*_csidap.log in the Resource directory.

How to enable the multitask SUPRA PDM

You can initiate the PDM manually or automatically. With automatic PDM initiation, the first task (either single-task running in concurrent mode or multitask) to attempt a database SINON initiates the PDM. Manual initiation means that you start the PDM from your login shell.

The UNIX user that starts a SUPRA PDM system must have adequate permissions to carry out the file- and system-resource operations the SUPRA PDM performs. The root user must start a systemwide SUPRA PDM system; a member of the group must start a groupwide SUPRA PDM system.

Manual PDM initiation

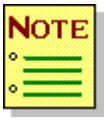
For manual PDM initiating, execute the pdm_startup script created by the s1_install script from the shell. Refer to the *SUPRA Server PDM UNIX Installation Guide,* P25-1008, for more details. Always initiate the PDM manually to test your setup before you rely on automatic startup. A manual startup will show any errors in your command file or input files, whereas an automatic startup cannot detect command file errors and gives unpredictable results. If the CSI_CONSOLE logical name has been defined, then additional information will be sent to the operator as an aid to debugging problems.

Automatic PDM initiation

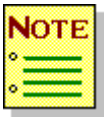
By default, the PDM is set to automatic initiation (meaning the first task to sign on to the PDM will automatically initiate it). However, always initiate the PDM manually to test your setup before you rely on automatic startup. Change the automatic default setting to NO by defining the logical name CSI_AUTOSTART = NO.

Automatic initiation requires that the daemons be up and all environment variables and logical names be defined. This can be done by running either daemon_startup or pdm_startup.

During automatic PDM initiation, the first task to attempt a database SINON is granted a PDM lock, which prevents other tasks from starting up the PDM on another machine. The task then communicates with the PDM either through shared memory (if the application and the PDM reside on the same machine) or through TCP/IP (if the application and the PDM reside on different machines). Once the PDM has started up on one of the machines on the preferred machine list, the PDM lock is released to allow other tasks to communicate with the PDM. See “Understanding automatic PDM startup” on page 111 for more information on automatic PDM startup.



If the PDM has exhausted all its startup attempts (you specify the number of times it may attempt to startup by defining the logical name CSI_START_RETRIES), it will fail with an error status of NMAC and will not attempt any additional restarts.



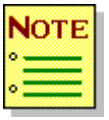
Do not use the automatic startup feature with systemwide SUPRA PDM systems. The automatic startup feature starts the SUPRA PDM system by the first user to perform a PDML function. If this user is not the root user, the SUPRA PDM system will not have adequate permissions to perform file and system resource operations, which can cause system calls to return permission-denied errors.

Logical names

The PDM relies on logical names to identify the various executable images and files it needs to run. Because logical names are not accessible across machines, include the definitions on each machine where a PDM may run.

You must make each logical name available to all user accounts that need to use it. Do this by placing the logical name definitions in each user's shell startup script or by placing the logical name in the group or system logical name tables. This process will make the logical name definitions available to all accounts with the same group ID or to the whole system. This process requires the definition only once.

Whether you start the PDM manually or automatically, you must define certain logical names. You need to decide where to best define these logical names. For example, you could define them as part of your UNIX system startup, or you could include them in a PDM-initiation script.



You must use a PDM-initiation script if you are starting up a remote PDM.

The following table lists the logical names that you must define, regardless of where you define them:

Logical name	Description
CSIPDMINP	Identifies the PDM input file.
6-character logical name for the database; may be 8, 9, or 10 characters, if prefixed.	Equates to the database description file.
*dbname*_CSI_PDM_MACS	Defines a preferred machine list for each database that will run in the PDM.
CSI_PDMID	Points to the name of the PDM to be used.
CSI_SYSPDMID	Points to the PDM name (if you are using the multiple systemwide PDM facility).

You may optionally define the logical name CSI_AUTOSTART. CSI_AUTOSTART identifies whether or not you want to start the PDM automatically. See “Implementing logical names” on page 38 for a complete list of logical names used by SUPRA PDM.

Example—enabling the PDM using an initiation script. The following information shows how to set up for PDM initiation by defining the required logical names in a PDM-initiation script. (“Understanding automatic PDM startup” on page 111 shows how to initiate the PDM using example initiation and input files.)

1. Create a SUPRA Server initiation script. This script does the following:

    a. Identifies the PDM input file with the logical name CSIPDMINP.

    b. Identifies the PDM output log file with the logical name CSIPDMLOG.

    c. Defines a 6-character logical name for each database description file. If prefixed, it defines an 8- to 10- character logical name for each prefixed, database description file.

    d. Defines a preferred machine list for each database that will run in the PDM.

    e. Optionally defines the logical name CSI_AUTOSTART to be NO if you wish to inhibit automatic PDM initiation.

    f. Optionally issues the command to initiate the PDM background process, if either manual startup or remote startup are required.

“How to enable the multitask SUPRA PDM” on page 82 describes the contents of a SUPRA Server initiation script and how to use a database prefix to allow users in the same group ID or system to access physically different databases with the same database name.

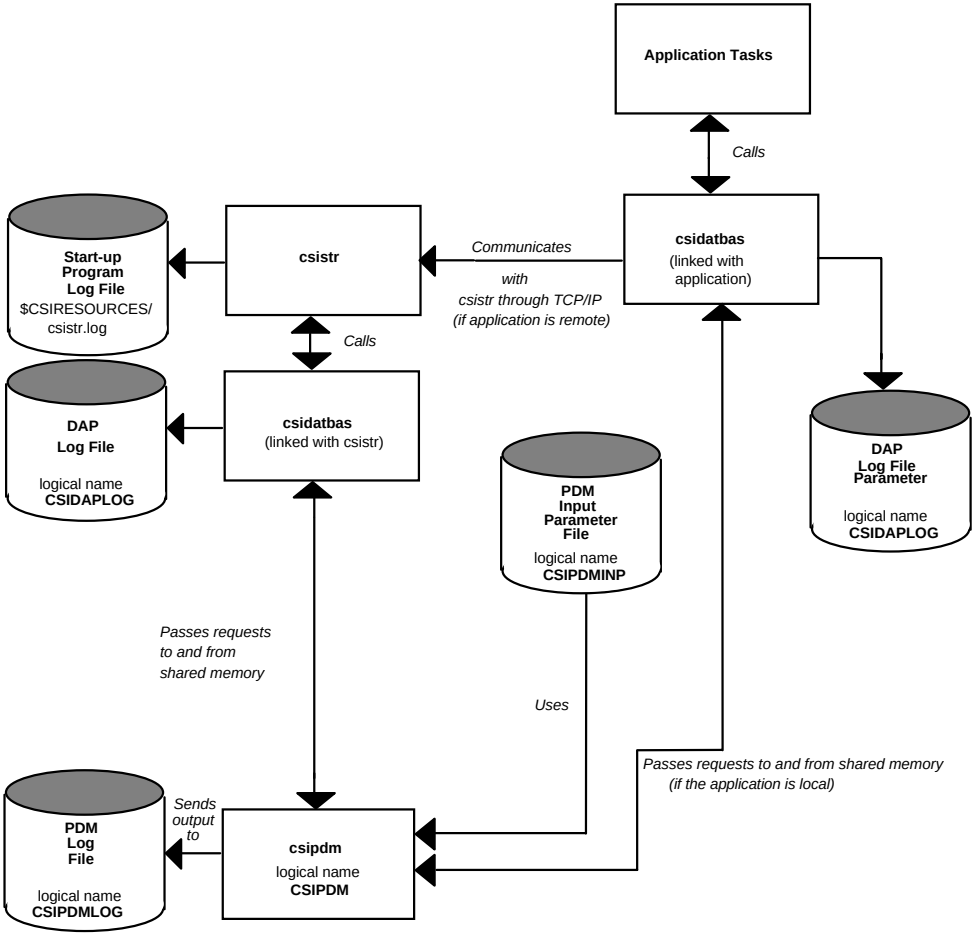
2. Create a PDM input file. This file contains parameters to define how the PDM is set up. The PDM locates the PDM input file by translating the logical name CSIPDMINP. “Using a database prefix” on page 94 describes the contents of the PDM input file.

3. Define the logical name CSI_PDMID to point to the name of the PDM to be used. This logical name must exist before PDM initiation, and it must be available to all user accounts that might execute an application that will use the PDM. See “Defining the logical name for the PDM (CSI_PDMID)” on page 108 for a description of the logical name CSI_PDMID.

4. To use the multiple systemwide PDM facility, define the logical name CSI_SYSPDMID to point to the PDM name. This logical name must exist before PDM initiation and must be available to each user who might execute an application that is to use the PDM (see “Defining the logical name for a multiple systemwide PDM (CSI_SYSPDMID)” on page 110).

The pdm_startup script created automatically by the install scripts for each SUPRA Server PDM system contains all of the above information except the individual database logical names. To define them, add csideflog entries to pdm_startup or another script. Then, execute the script.

The following figure shows what files and images are used during PDM initiation:



Creating a PDM initiation script

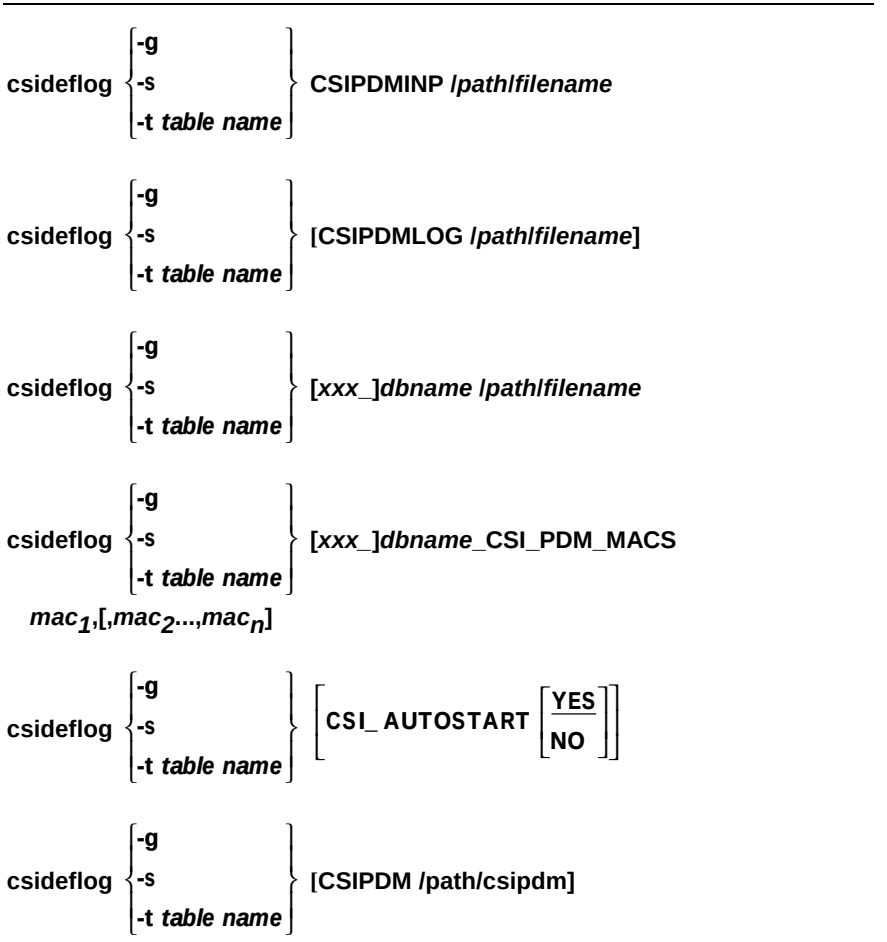
To enable manual and automatic PDM initiation, use a standard text editor such as vi or emacs to edit the pdm_startup script for the PDM system you want to run.

This example shows the additional lines that must be added to the pdm_startup script to start the PDM on machine UNIXA, UNIXB, or UNIXC for TESTDB.

```
# Group-level logical names for the three TESTDB
# compiled database descriptions
#
  csideflog -g TESTDB /home/data1/testdb.mod
  csideflog -g PRD_TESTDB /home/data2/testdb.mod
  csideflog -g DEM_TESTDB /home/demo/testdb.mod
#
  csideflog -g TESTDB_CSI_PDM_MACS       UNIXA,UNIXC
  csideflog -g PRD_TESTDB_CSI_PDM_MACS   UNIXC,UNIXB
  csideflog -g DEM_TESTDB_CSI_PDM_MACS   UNIXA
```

Each script contains the csideflog utility to create the necessary logical names, and the csipdm executable image to initiate the PDM process in background. The name for the PDM is constructed from the logical name CSI_PDMID and the group ID in which the PDM is running (zeros if systemwide). When using the multiple systemwide PDM facility, the name for the PDM is constructed from the logical name CSI_SYSPDMID and six zeros. In both cases, this name is used to construct shared memory segments.

You create a PDM initiation script in the following format:



-g
-s
-t *table name*

Description *Required*. Specifies the logical name table in which to place the specified parameters.

Options

-g	Group logical name table
-s	System logical name table
-t *table name*	User logical name table

CSIPDMINP */path/filename*

Description *Required*. Identifies the file containing the PDM input parameters.

Format Valid UNIX file specification

Considerations

- Create the input file using a UNIX text editor.
- The file to which this logical name points must be unique for each PDM you wish to start.

CSIPDMLOG */path/filename*

Description *Optional*. Identifies the output file to which the PDM will send all messages.

Format Valid UNIX file specification

Considerations

- If you do not define the logical name CSIPDMLOG, the PDM will create a file called CSIPDMLOG in your default directory.
- The file to which this logical name points must be unique for each PDM you wish to start.

[*xxx*_]*dbname /path/filename*

Description *Required*. Assigns a logical name to the compiled database description file and places this name in either the group, system or the user logical name table.

Format

[*xxx*_]	(*Optional*) 1- to 3-character database prefix followed by an underscore
dbname	6-character database name
/*path*/*filename*	Valid UNIX file specification for compiled database description file

Considerations

- Define a separate logical name for each database you wish to use.
- You must define this logical name before you can format the physical files for the database.
- Groupwide databases may only be accessed by users in the same group as the initiating task.
- Systemwide databases may be accessed by all users of the system.
- You may prefer to make the logical assignment for the compiled database description elsewhere either manually or at system startup. In this case, it is not needed in the PDM initiation script. Because this logical name must exist somewhere on the system, you may prefer to duplicate it in your PDM initiation script rather than to risk omitting it.
- The 1- to 3-character prefix allows you to differentiate between databases of the same name that run in the same PDM. See "Creating a PDM initiation script" on page 88 for a description of how to use a database prefix.
- The file to which this logical name points must be unique for each PDM you wish to start.

[*xxx*_]*dbname*_CSI_PDM_MACS *mac*$_1$,[,*mac*$_2$...,*mac*$_n$]

Description *Required*. Specifies a list of machines that can be used on the specified database, in order of preference (the preferred machine list). "Understanding automatic PDM startup" on page 111 describes a sample network environment.

Format

[*xxx*_]	(*Optional*) 1- to 3-character database prefix followed by an underscore
dbname	6-character database name forming the variable part of the logical name
mac$_1$	Node name of the first choice machine on which the database can be used
[,*mac*$_2$...,*macn*]	Node names of any other machines on which the database can be used, in order of preference

Considerations

- Create a preferred machine list for each database to be used.
- If the preferred machine list contains more than one machine, then duplicate the logical assignments on each machine that might execute an application.
- The logical name *dbname*_CSI_PDM_MACS must be accessible to all applications that might access the specified database.
- You may prefer to make these logical assignments elsewhere, either manually or at system startup. In this case, they are not needed in the PDM initiation script. Because these logical names must exist somewhere on the system, you may prefer to duplicate them in your PDM initiation script rather than to risk omitting them.
- The PDM can load a database and access its files if they reside on a different node, provided they are accessible to it (on an NFS-mounted file system). Files made available this way, however, cannot reside in a SUPRA Server file system and can only be accessed using standard UNIX I/O functions. NFS-file access may not be reliable, so it is advised to keep everything local where possible.
- The 1- to 3-character prefix allows you to specify a different preferred machine list for databases with the same name in the same PDM. See "Creating a PDM initiation script" on page 88 for a description of how to use a database prefix.

CSI_AUTOSTART [**YES** / **NO**]

Description *Optional*. Enables or disables the automatic PDM initiation facility.

Default YES

CSIPDM /path/csipdm

Description *Conditional*. Required when CSI_AUTOSTART is enabled. Specifies the path to the csipdm executable image.

Format A valid UNIX path for the csipdm executable image

Consideration This logical name is used by CSIDATBAS to locate the csipdm image during autostart operations.

Using a database prefix

A database prefix allows you to distinguish between databases of the same name that run within the same group or system. This section describes how to implement a database prefix for groupwide databases. To apply a database prefix to systemwide databases, substitute -s for -g in the logical definitions described below.

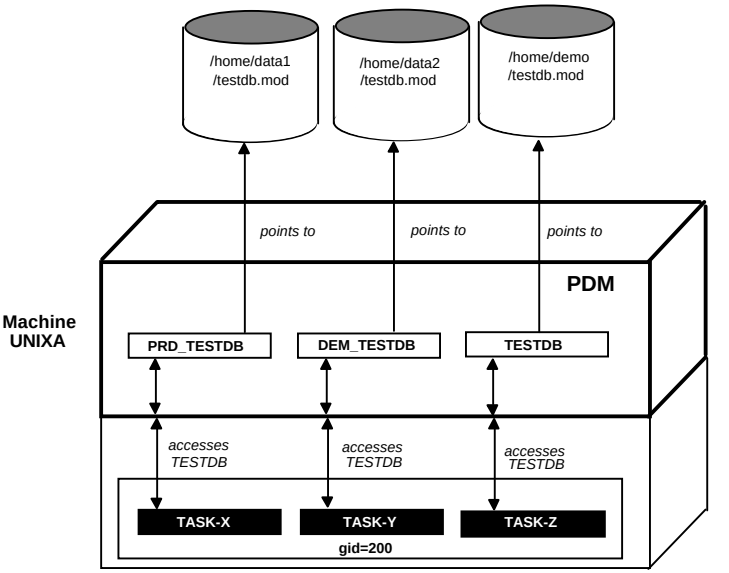
For example, assume that you have three databases called TESTDB, all used by tasks with group ID 200 on machine UNIXA. Each database uses a different compiled database description. You can have only one group-level logical name: TESTDB, which points to one compiled database description. The database prefix allows you to distinguish the other two by creating a group-level logical name *xxx*_TESTDB for each database, where *xxx* is a unique 1- to 3-character identifier, for example,

```
csideflog -g xxx_TESTDB.
```

The following figure shows three tasks (TASK-X, TASK-Y, and TASK-Z) running in group ID 200 on machine UNIXA. They all sign on to a TESTDB database using the logical name TESTDB. Each task, however, uses a different compiled database description. To identify the compiled database descriptions they are accessing, each task defines the process logical name CSI_PREFIX; for example,

```
csideflog -p CSI_PREFIX xxx
```

where *xxx* is the 1- to 3-character identifier to be used as a prefix to the logical database name used.



Considerations for programs that use database prefixes

- Each task can define only one value for CSI_PREFIX and can redefine it to access a different compiled database description. Any task that omits the logical definition for CSI_PREFIX accesses the default database identified by the unprefixed logical name TESTDB.
- When a task accesses a database, the PDM first attempts to translate the prefixed database name and then the unprefixed database name to find the default compiled database description. Therefore, it is important to define the unprefixed, 6-character logical name (TESTDB, in this case) pointing to the default compiled database description.
- When you set up a database prefix, consider that if one task loads a database without a prefix, a subsequent task can define a prefix for that database and sign on using that prefix. If this happens, the database is now associated with that prefix, and subsequent tasks must use the prefixed sign-on name. The first task, however, can still use the loaded database and sign off without problems.

Once a task has loaded a database with a prefix, it cannot change or remove the prefix without first unloading the database.

- Each database must have the following logical names defined for each prefix:

  *xxx*_dbmod /path/dbmod.mod

  *xxx*_dbmod_CSIPDM_MACS Machine list

  *xxx*_path Path to database files

  .

  .

  .

  DUMPSLF_*xxx*_dbmod dumpslf input file

Creating a PDM input file

You identify the PDM input file by defining the group or system-level logical name CSIPDMINP in the PDM initiation script. The following shows the parameters in the input file:

[BATCHTHREADS=*nnnnn*]
[CONSOLE = { Y / N }]
[DYNSLOCK = { Y / N }]
[HASHCNT=*nnn*]
[INTERVAL=*nnnn*]
[MAXDATA=*nnnnn*]
[MAXTASKS=*nnnn*]
[MAXTHREADS=*nnn*]
[MRELAY = { Y / N }]
[MULTIHOLD = { Y / N }]
[OPERATOR = { OPER*nn* / OPER1 }]
[PDMNAME=*pdmname*]
[PROTCHECK = { Y / N }]
[READAHEAD_THRESHOLD1=*nnnnn*]
[READAHEAD_THRESHOLD2=*nnnnn*]
[RETRY=*nnn*]
[SIGNAL_TRAP = { Y / N }]
[STATISTICS = { Y / N }]
[SYSOPCOM = { Y / N }]
[TIMEOUT=*nnnnn*]

BATCHTHREADS=*nnn*

Description *Optional*. Specifies the maximum number of concurrent single-task PDM applications that the PDM can have per database.

Default 0

Options 1–100

CONSOLE= {**Y** | **N**}

Description *Optional*. Specifies whether to display all PDM messages sent to an operator terminal.

Default Y

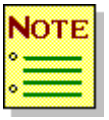
Options

Y Displays all PDM messages.

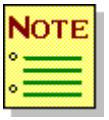
N Suppresses all PDM messages except “Reply with a PDM command” so you can communicate with the PDM via the operator console.

DYNSLOCK= { **Y** / **N** }

Description *Optional*. Specifies whether you want the data sets in a database to remain locked if they have been updated by a task that fails to sign off normally when no task logging is in use.



Warning: Use of the DISABLE/DYNAMIC operator command overrides this option (if you issue DISABLE/DYNAMIC for the database, then the files will be unlocked regardless of the dynslock setting, and the database will be disabled). See "Disabling a database (DISABLE)" on page 127 for a complete description of DISABLE/DYNAMIC.



Warning: If a task fails to sign off normally when data sets have been updated and there is no active task log, the updated data sets may be logically incorrect.

Default Y

Options

Y Data sets are to remain locked if the task fails to sign off normally and updates have been made when no task log is active.

N Data sets are not to remain locked if the task fails to sign off normally and updates have been made and no task log is active.

Considerations

- Setting the parameter value to y keeps file locking consistent with earlier releases of the PDM.
- If the PDM fails and there is no active task log, the updated data sets will remain locked.

HASHCNT=*nnn*

Description *Optional.* Specifies the number of buffers to search in the buffer pool for the requested block. This number also determines the type of buffer access used. Buffer pools containing less than hashcnt buffers will be searched sequentially. Buffer pools containing equal to or more than hashcnt buffers will be searched using a hashing algorithm.

Default 20

Options 10–999

Considerations

- This parameter should be changed only after consulting Cincom Support.
- For buffer pools containing very large numbers of buffers, a performance improvement may be gained by increasing hashcnt.

INTERVAL=*nnnn*

Description *Optional*. Specifies the period between the PDM's attempts to obtain a held record (measured in seconds).

Default 5

Options 1–1000

Consideration Used by the TIMEOUT and RETRY parameters.

MAXDATA=*nnnnn*

Description *Optional*. Sets the maximum size of the message buffer used for communication between the PDM and applications.

Default 4096

Options 0–32,767

Considerations

- The message area must be large enough to contain all of the parameters passed to the PDM. These include the following:
  - User data area
  - Element list
  - Key field
  - Status function
  - End parameters
- The value of MAXDATA * MAXTASKS is roughly equivalent to the size of one of the PDM's shared memory segments.

MAXTASKS=*nnnn*

Description *Optional*. Specifies the maximum number of allowed concurrent accesses to the PDM.

Default 50

Options 1–32766

Considerations

- You may wish to impose a maximum in order to ensure good performance for tasks that access the PDM.
- The value of MAXDATA * MAXTASKS is roughly equivalent to the size of one of the PDM's shared memory segments. Therefore, you may need to adjust the system parameter for maximum shared memory size if you use large values.

MAXTHREADS=*nnn*

Description *Optional*. Specifies the maximum number of threads that the PDM can have per database.

Default 3

Options 1–100

Considerations

- A thread is a function processor; for example, a value of 3 means three concurrent functions.
- The optimum value for MAXTHREADS is the maximum number of concurrent functions that could be issued to the PDM at a given instance plus two. The more threads you have, the more concurrent I/Os you can have. The optimum value will vary from site to site. A process is created for each thread.
- Specifying a low value for MAXTHREADS prevents database concurrence. Specifying a high value for MAXTHREADS will use more system resources and create more processing overhead. It is better to specify a high value than a low value because too few overlapping threads can inhibit performance.

MRELAY = 

Description *Optional*. Specifies whether to send messages output by the PDM and the system log dump program, cstudslf, to a named pipe.

Default N

Considerations

- If you specify Y, users must then write programs to pick up console messages from the named pipe. If the named pipe fills up, the PDM keeps a count of the number of messages lost and displays this number on the next successful send.
- You can use MRELAY in conjunction with the logical name CSI_MRELAY to trap all messages from SUPRA Server components. See “Writing your own interface to SUPRA Server PDM” on page 154 for more details about writing a named pipe reading program to read the PDM messages.

MULTIHOLD = { Y / N }

Description *Optional*. Specifies whether a task can explicitly read and hold more than one record per file in a single COMIT period.

Default Y

OPERATOR= { OPER*nn* / OPER1 }

Description *Optional*. Identifies the PDM operator terminals.

Default OPER1

Options OPER1–OPER12

Considerations

- You can specify only one operator terminal in the input file, although you can enable more than one terminal as that operator number.
- The PDM prompts the specified operator terminal at regular intervals. Use the csireply utility to respond to the operator prompt.

PDMNAME=*pdmname*

Description *Conditional*. Required only if using the multiple systemwide PDM facility. Specifies the name for the systemwide PDM.

Format 1–8 alphanumeric characters

Considerations

- Do not use the pdmname parameter if you are not using the multiple systemwide PDM facility.
- The PDMNAME must be the same as the equivalence string for the logical name CSI_SYSPDMID.

PROTCHECK={**Y** | **N**}

Description *Optional*. Causes the PDM to check the permissions set on data files before allowing a task to access them.

Default Y

READAHEAD_THRESHOLD1=*nnnnn*

Description *Optional*. The number of requests at which the PDM will begin to pre-read (or readahead) records from the same file with the same parameters.

Default 10

Options 1–32,766

Considerations

- For RDNXT, READV and READR, the PDM will only do logical I/Os (I/Os with no physical access to the disk) to fill the buffer. For READX, the PDM will perform up to 10 reads to fill the buffer. These may be logical or physical I/Os.
- This option is global—in other words, it applies to all files. Be sure to consider all possible conditions under which files will be accessed.
- Lowering this threshold will cause readahead to engage sooner, which can cause performance improvement under some conditions, but can cause problems if the number of requests from the application is at or close to the threshold.
- Buffer size can be controlled with the MAXDATA parameter.

READAHEAD_THRESHOLD2=*nnnnn*

Description *Optional*. Number of requests of data from the same file before the PDM begins to do physical I/O to fill the buffers and satisfy the readahead.

Default 100

Options 1–32,766

Considerations

- Must be a number between READAHEAD_THRESHOLD1 and 32766.
- This option is global—in other words, it applies to all files. Be sure to consider all possible conditions under which files will be accessed.
- Lowering this threshold will cause readahead to engage sooner, which can cause performance improvement under some conditions, but can cause problems if the number of requests from the application is at or close to the threshold.
- Buffer size can be controlled with the MAXDATA parameter.

RETRY=*nnn*

Description *Optional*. Specifies the number of times the PDM will attempt to obtain a held record before returning a HELD status.

Default 5

Options 1–100

Consideration The PDM waits for the period specified in INTERVAL before retrying. For example, INTERVAL = 5 and RETRY = 10 cause a PDM to retry ten times at five-second intervals. The total elapsed time would be 50 seconds.

SIGNAL_TRAP= {**Y**, **N**}

Description *Optional*. Causes the PDM to set up only the required signal traps to operate. If an untrapped signal occurs, the PDM will abort with a core dump.

Default Y

Options

Y Enable signal trapping

N Disable signal trapping

STATISTICS= {**Y**, **N**}

Description *Optional*. Indicates whether you want detailed PDM statistics.

Default N

Options

Y Collect statistics

N Discard statistics

Considerations

- Y sends the statistics to the following:
  - The PDM log file as identified by the logical name CSIPDMLOG in the PDM initiation script. "How to enable the multitask SUPRA PDM" on page 82 describes the PDM initiation script.
  - The named pipe for message reading if MRELAY = Y.
  - The csiopcom utility. "Communicating with the SUPRA Server PDM" on page 119 describes the csiopcom utility to PDM.
- See "PDM statistics output" on page 257 for an explanation of the statistics output.

SYSOPCOM= {**Y**, **N**}

Description *Optional*. Inhibits the display of the csireply prompt “Reply with a PDM command” at PDM operator devices.

Default Y

Options

Y Enables communication with the PDM via both the csireply utility and the csiopcom utility

N Enables communication only through csiopcom

TIMEOUT=*nnnnn*

Description *Optional*. Specifies the period in number of INTERVALS before a task is dynamically signed off if it remains inactive.

Default 0 (None, it can stay forever)

Options 0–10,000

Example This example illustrates the contents of a PDM input file, specifying values for all possible parameters:

```
CONSOLE=N
INTERVAL=25
MAXDATA=16384
MAXTASKS=10
MAXTHREADS=7
MRELAY=N
MULTIHOLD=Y
OPERATOR=OPER9
PDMNAME=CINCOM
PROTCHECK=Y
RETRY=10
SIGNAL_TRAP=Y
STATISTICS=Y
SYSOPCOM=N
TIMEOUT=2400
```

Enabling/disabling automatic PDM startup (CSI_AUTOSTART)

You can enable or disable automatic PDM initiation by defining the logical name CSI_AUTOSTART in this format:

csideflog -g CSI_AUTOSTART { **yes** / **no** }

{ **yes** / **no** }

Description *Required*. Specifies whether automatic PDM initiation is enabled or disabled.

Default yes

Consideration When CSI_AUTOSTART is enabled, the logical name CSIPDM must be defined with the path to the csipdm executable image.

Defining the logical name for the PDM (CSI_PDMID)

CSI_PDMID equates to the 1- to 8-character name of the PDM and must be defined before PDM initiation. Place the logical definition in the group or system logical name table according to whether the PDM is groupwide or systemwide. The format for defining CSI_PDMID is as follows:

csideflog { **-g** / **-s** } **CSI_PDMID** ***pdmname***

{ **-g** / **-s** }

Description *Required*. Specifies whether the PDM runs groupwide or systemwide.

Options

-g Groupwide

-s Systemwide

Considerations

- A groupwide PDM may be accessed only by users with the same group ID as the initiating task.
- A systemwide PDM may be accessed by all users on the system.

pdmname

Description *Required*. Specifies the name of the PDM.

Format 1- to 8-character alphanumeric name

Considerations

- csidatbas needs to access CSI_PDMID to allow local automatic PDM startup.
- csistr needs to access CSI_PDMID to allow remote automatic PDM startup.

General consideration

You can make sure that the first PDM initiation is executed manually by including the logical definition for CSI_PDMID in the PDM initiation script only. Once you execute the PDM initiation script, CSI_PDMID remains in the group or system logical name table until the machine goes down or the CSI_PDMID is explicitly deleted. After the machine is rebooted, you will have to restart the PDM manually.

Alternatively, if you want to allow automatic PDM startup even after a machine close-down, make sure that CSI_PDMID and the other logical names exist before the first application task attempts to sign-on to the PDM. You can do this by including the logical name definitions as part of your system startup.

Defining the logical name for a multiple systemwide PDM (CSI_SYSPDMID)

CSI_SYSPDMID equates to the 1- to 8-character name of a multiple systemwide PDM. Place the logical name in the process table for each process that will use a multiple systemwide PDM. The format for defining CSI_SYSPDMID is as follows:

csideflog CSI_SYSPDMID *pdmname*

pdmname

Description *Required*. Specifies the name of the multiple systemwide PDM.

Format 1–8 alphanumeric characters

Considerations

- Use CSI_SYSPDMID only if you wish to have multiple systemwide PDMs running.
- Place the logical name in the process table for each process that will use a multiple systemwide PDM.
- CSI_SYSPDMID takes precedence over CSI_PDMID.
- CSIDATBAS needs access to CSI_SYSPDMID to allow automatic PDM startup.
- The PDM input parameter file (pointed to by the logical name CSIPDMINP) must contain the PDMNAME = *xxxxxx* entry, where *xxxxxx* must be the same as the definition for CSI_SYSPDMID.
- All logical names that the systemwide PDM needs to use (dbmods, preferred machine lists, files, etc.) must be defined in a user logical name table. The name of this table must be of the form CSI_PDM_*xxxxxxx*, where *xxxxxxx* must be the same as the definition for CSI_SYSPDMID.
- A systemwide PDM may use groupwide and systemwide databases. The PDM may be accessed by all users on the system.

Understanding automatic PDM startup

Automatic PDM initiation starts up a PDM without any user intervention. It can be used on the first PDM startup as well as to start the PDM again after a failure. It is initiated and controlled by the first application task that tries to access a database owned by the PDM.

If the PDM fails due to a hardware, software, or communication fault, the first task to attempt a database access on the failed PDM restarts it. The PDM could fail for any of the following reasons:

- Hardware fault (a machine check)
- Software fault (a PDM abort)
- Communication fault (a network error)

The initiating task uses the logical name CSIPDM to locate the csipdm binary image that will be executed. This logical name should point to one of the following:

- The production binary (csipdm) located in the install directory under bin.
- The debug binary (csipdm_debug) located in the install directory under dbin.

The initiating task sets up a lock on the PDM preventing other tasks from restarting the PDM for the same database. Automatic PDM initiation uses the preferred machine list to identify alternative machines on which the PDM may run.

Once the PDM has restarted, the lock is released. Other tasks then execute a dynamic SINON (handled by CSIDATBAS) which makes the PDM perform a warm start on the database. When the warm start is complete, the PDM returns a status of DRST (dynamic reset) to each task. The tasks must then reapply any modifications made since the last successful COMIT point.

Any errors that occur in the startup process (prior to the opening of the log file by the PDM) are written to a file in the $CSIRESOURCES directory. The name of the file is composed as the following example demonstrates:

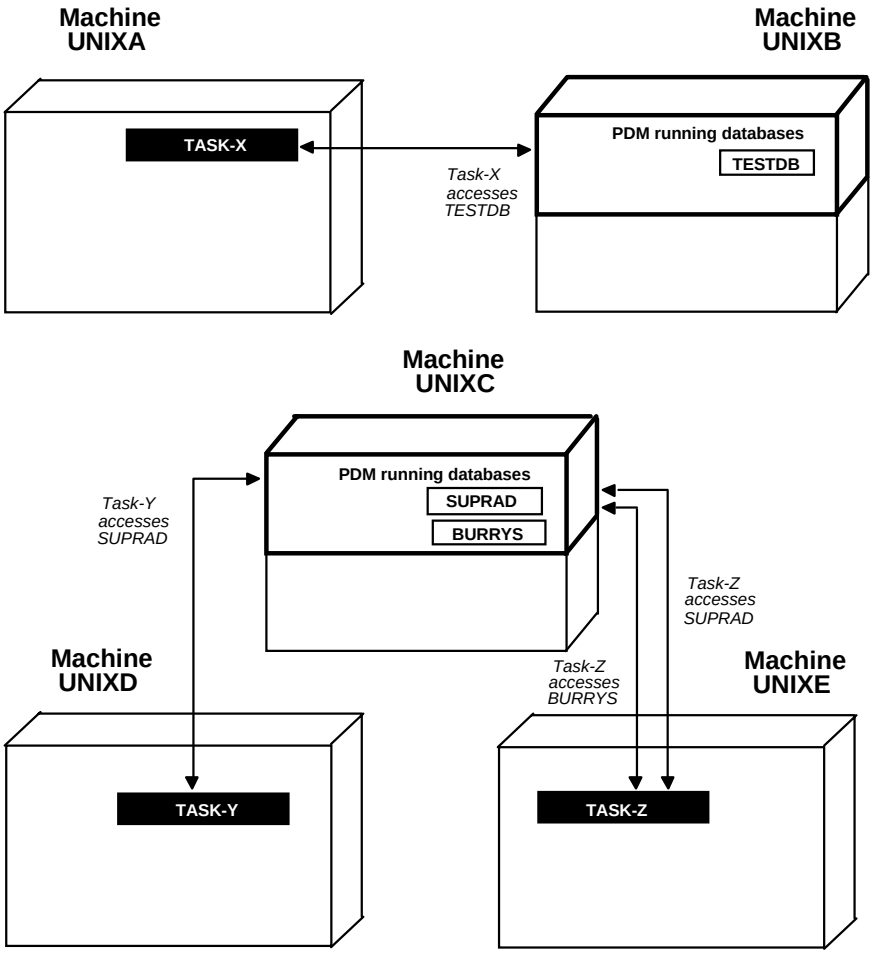
$CSIRESOURCES/CSI_PDMID_ [**GROUP** | **SYSTEM** | **MULTI_SYSTEM**] **_WIDE.err**

This file may contain one of the following errors:

```
csipdm: CSIRESOURCES not defined
csipdm: Could not set CSIPID
csipdm: Could not set user id to root
csipdm: Tidy up daemon CSITIDY not running -
                Cannot Continue
csipdm: Error opening log file ______
csipdm: Error opening input file _______
```

The following figure shows an example of a network configuration consisting of five machines, UNIXA through UNIXE. Three databases, SUPRAD, TESTDB, and BURRYS are active. TESTDB runs in one copy of the PDM on machine UNIXB; SUPRAD and BURRYS run in another copy of the PDM on machine UNIXC. The following tasks access these databases:

- TASK-X on machine UNIXA
- TASK-Y on machine UNIXD
- TASK-Z on machine UNIXE



Both copies of the PDM share the same name. The preferred machine list logical definitions for the three databases are as follows:

```
csideflog -g TESTDB_CSI_PDM_MACS  UNIXB,UNIXC
csideflog -g SUPRAD_CSI_PDM_MACS  UNIXC,UNIXA,UNIXB,UNIXD,UNIXE
csideflog -g BURRYS_CSI_PDM_MACS  UNIXC,UNIXE
```

All three databases have been initiated on the first machine in their preferred machine list.

Assume machine UNIXB fails. The database TESTDB running on machine UNIXB is no longer available. TASK-X, which is using database TESTDB from machine UNIXA, will receive a message to say that its PDM has gone down. The first database access after the PDM failure will do the following:

1. Look up the first machine on the machine list identified by the logical name TESTDB_CSI_PDM_MACS. This machine is UNIXB.

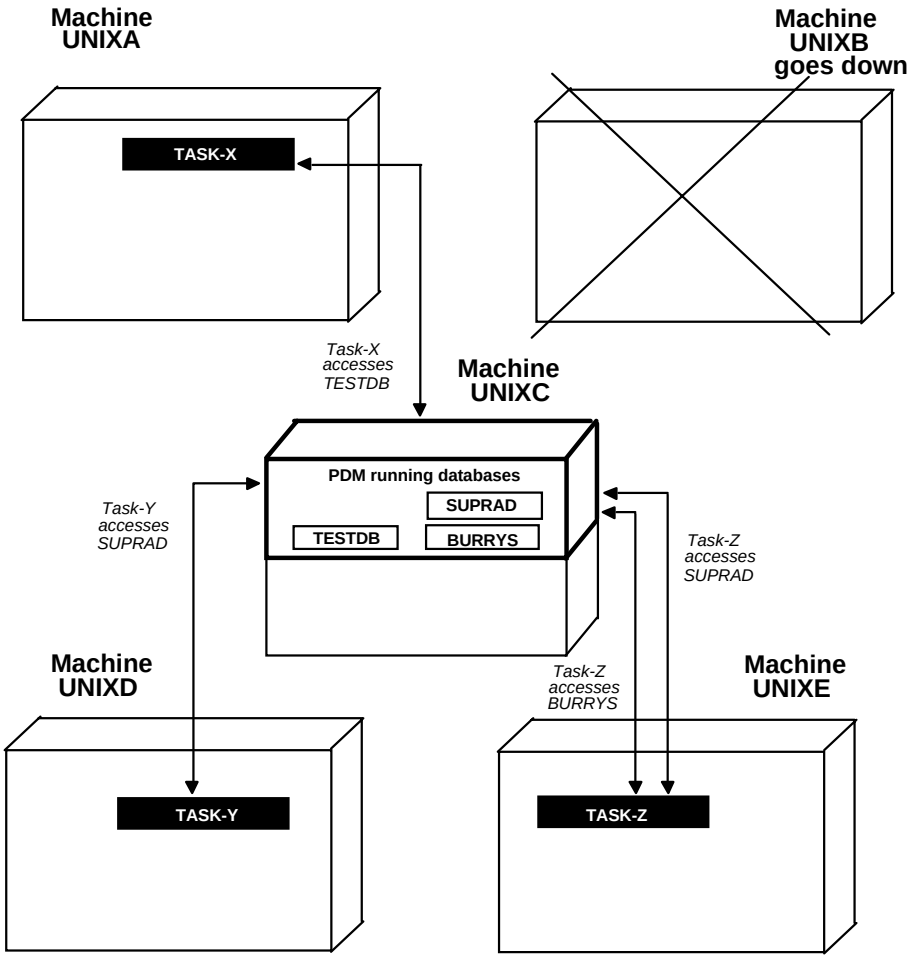
2. Discover that UNIXB is unavailable since no started program will respond.

3. Look up the next machine on the machine list identified by the logical name TESTDB_CSI_PDM_MACS. This machine is UNIXC.

4. Discover PDM is already running on UNIXC through the starter program (csistr).

5. Establish communications with the PDM.

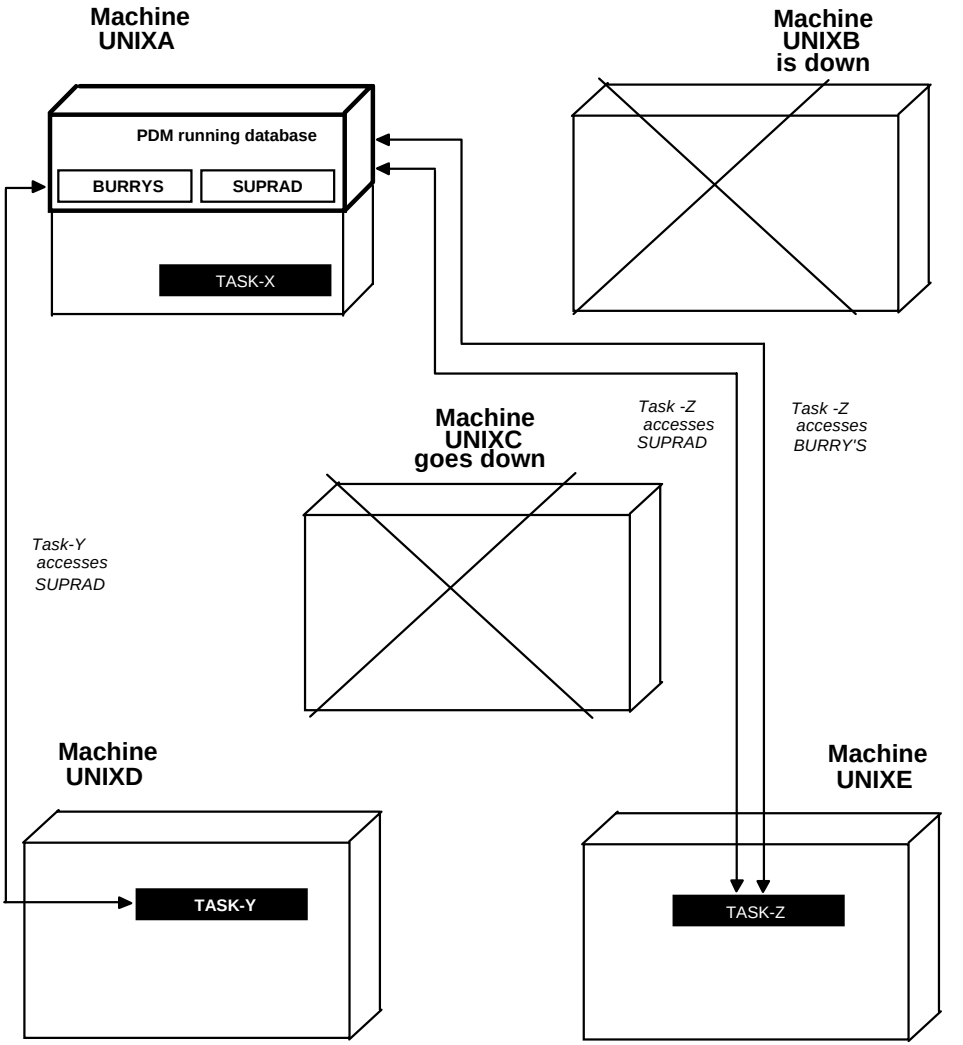
The following figure illustrates the network configuration after the PDM completes Step 5.



If machine UNIXC now fails, all tasks connected to it are disconnected. The next task to access SUPRAD starts the PDM on UNIXA (the next machine in the preferred machine list for SUPRAD). The PDM is then local for TASK-X, also running on UNIXA. TASK-Y on UNIXD and TASK-Z on UNIXE will connect to UNIXA when they next access SUPRAD.

Any application task that tries to use TESTDB receives a NMAC (No machine) status code because the preferred machine list (UNIXB, UNIXC) has been exhausted. However, if either UNIXB or UNIXC becomes available before a task attempts to access TESTDB, the PDM for that database restarts there.

The next task to access BURRYS starts it on UNIXE, which is the next machine on the BURRYS preferred machine list. The following figure illustrates the new machine configuration.



Single-task PDM

The single-task SUPRA PDM runs in the application process. The application is linked with csibatbas.o instead of csidatbas.o. csibatbas.o contains all of the logic to perform PDML functions. This makes single-task SUPRA PDM applications somewhat larger than multitask SUPRA PDM applications. The benefit of single-task SUPRA PDM is increased performance, especially in a stand-alone batch environment.

The application task and the PDM logic communicate through the database access program, csibatbas. The application programmer codes PDML calls identical to the PDML calls for multitask SUPRA PDM, using the entry point DATBAS. These calls are transferred to the SUPRA PDM logic through direct calls. When processing is complete, SUPRA PDM passes the results of the functions directly to the application program.

When the application program executes the SINON PDML function, it initiates the single-task SUPRA PDM. The SINOF PDML function deinitializes the single-task SUPRA PDM system. There are two modes of operation for single-task SUPRA PDM, concurrent and stand-alone.

Concurrent mode

In concurrent mode, the multitask PDM is required to be running prior to executing a single-task application. The multitask PDM may be brought up by using the pdm_startup script or the single-task PDM will bring it up automatically if the CSI_AUTOSTART logical name is set to YES. The PDM input parameter BATCHTHREADS must be set to 1 or more for single-task applications to operate. (A BFUL error will result if the BATCHTHREADS are exhausted.) In concurrent mode, system logging is fully operational for both single-task and multitask SUPRA PDM applications.

When running in concurrent mode, multiple single-task SUPRA PDMs may access the same database files in UPDATE mode concurrent with a multitask SUPRA PDM accessing the database files in UPDATE mode. This is possible because both single-task and multitask SUPRA PDM share the same copy of the loaded dbmod, which contains the record holding table and the buffer pools. This mode of operation is only available on the HP-UX platform.

Stand-alone mode

In stand-alone mode, the multitask SUPRA PDM and all other single-task applications must be shut down prior to executing a single-task application. This was the standard operation of single-task SUPRA PDM in prior releases and is now the default operating mode. In stand-alone mode, it is not possible to run system logging. Task logging and recovery, however, are fully operational in both concurrent and stand-alone modes of operation.

The mode of operation of single-task SUPRA PDM depends on the platform and the value of the logical name CSI_BATCH_CONCURRENT. The stand-alone mode is the only mode available for the NCR3000, AIX, and Digital UNIX platforms. For the HP-UX platform, the concurrent mode of operation is optional. On this platforms it is possible to run single-task applications in concurrent mode by defining the logical name CSI_BATCH_CONCURRENT as YES. By default the mode of operation is stand-alone.

5

Communicating with the SUPRA Server PDM

SUPRA Server PDM runs as a background process. Because of this, you can communicate with the PDM using a set of PDM operator commands. These operator commands allow the following controls:

- Controlled shutdown of the SUPRA Server PDM
- Temporary disabling of the SUPRA Server PDM
- Display of loaded databases, tasks, and statistics
- Disabling, unloading, and mode changing of databases
- Invocation of system-log dumping of databases

Entering PDM operator commands

You enter PDM operator commands in one of two ways:

- **Using csiopcom.** A screen based utility which allows commands to be typed at a command line or selected through a menu. csiopcom also provides comprehensive online Help. Commands given through the csiopcom application are sent to the PDM using the OPCOM DML function (through DATBAS). Therefore, it is only possible to send commands to a SUPRA Server PDM which is accessible to your effective group ID.

  csiopcom does allow commands to be sent to PDMs running on other nodes in your network with the same group rules applying. The default is set to allow informational commands only (display commands) for all users except the supervisor. However, you have a facility that allows you to create authorization files on a per user basis if you wish for some users to have access to some of the potentially more dangerous commands. See "Communicating with SUPRA Server PDM using csiopcom" on page 148 for additional information about csiopcom.

  You use the csioauth utility to create authorization files. The csioauth utility is a screen-based program that operates through a series of YES/NO questions and creates an authorization file. The utility can only be run by users possessing the DBAPRV privilege. See "Understanding user privileges" on page 57 for a discussion on privileges. This utility prevents users from creating privilege files of their own. When csiopcom is invoked, the privilege file is picked up through the logical name CSIOPCOM_AUTH.

- **Using csireply.** A command line application run from the shell. This application allows PDM commands to be sent to ANY PDM on the node from which the command is issued. Using csireply, any operator command can be issued to the PDM. The csireply command can only be run by users possessing the REPLY privilege. See "Understanding user privileges" on page 57 for a discussion of privileges.

Eleven PDM operator commands are available; however, you may not wish to give all users access to all these commands. If you are communicating with the PDM through csiopcom, you can use the user authorization program csioauth to restrict a set of commands to an individual user. The csioauth program can create a user authorization file that is used by the csiopcom program through the logical name CSIOPCOM_AUTH.

Users of the csireply command can use any of the 11 PDM operator commands. To communicate with the PDM using the csireply command, see “Communicating with the PDM using the csireply command” on page 160.

The user authorization file applies only to users communicating with the PDM through csiopcom.

You can also write your own interface to SUPRA Server PDM. “Writing your own interface to SUPRA Server PDM” on page 154 describes how to direct the SUPRA Server PDM messages to a named pipe that can be read by a user-written program. The interface provides an example program in pseudocode to read from the named pipe. See “Example mailbox read program” on page 263 for an example of a named-pipe-reading program written in C programming language.

Controlling the PDM with operator commands

The SUPRA Server PDM operator commands control PDM usage. The PDM recognizes the following operator commands:

Command	Description
ACTIVATE	Enables the use of an index.
DEACTIVATE	Disables the use of an index.
DISABLE	Unloads one or all database(s) from the PDM and prevents anyone else from reloading.
DISPLAY	Displays the status of loaded databases, the status of signed-on tasks, or statistics.
DUMPSLF	Dumps a system log component manually.
ENABLE	Cancels the DISABLE restriction.
POPULATE	Rebuilds an index and activates it.
PRINT	Produces a file that is defined by the logical name CSI_DMPANL.
READONLY	Sets one or all database(s) to read-only processing.
SHUTDOWN	Unloads all databases and shuts down the PDM.
UNLOAD	Unloads one or all databases from the PDM.
UPDATE	Cancels the READONLY restriction.

Activating an index (ACTIVATE)

The ACTIVATE command enables the use of an index as well as the automatic updating of the index by SUPRA Server PDM. Each change to the data file is reflected by corresponding changes to the index file(s). Changes to the index file include updating, adding, or deleting index records.

ACTIVATE *index* [*xxx*_]*database-name* [***[group - name]***]

[AT *node-name*]

index

Description *Required*. Specifies the index file to be activated for logging.

Format *dset*IX*yy*

where: *dset* = 4-character data set name

IX = entered as shown

yy = 2-character index name

Consideration If the index has been deactivated for any length of time, run the check option of the index maintenance utility program identified by the logical CSTUIDX. This checks the index records and updates them where necessary, and automatically activates the index.

[*xxx*_]*database-name*

Description *Required*. Identifies the database for which index logging is to take effect.

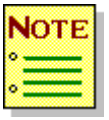
Format

*xxx*_	1–3 alphanumeric characters followed by "_" (optional database prefix)
database-name	6 alphanumeric characters

Consideration If the database is prefixed, the 6-character database name will be preceded by the 1- to 3-character prefix name and an underscore. If no prefix is used, the database name will always be 6-characters.

[***[group - name]***]

Description *Optional*. Identifies the group if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters

AT *node-name*

Description *Optional*. Directs the command to the remote PDM running at the specified node.

Format 1–6 alphanumeric characters

Considerations

- If you omit this parameter, the command is directed to the local PDM.
- Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Deactivating an index (DEACTIVATE)

The DEACTIVATE command disables the use of an index as well as the automatic updating of the index by SUPRA Server PDM. This means the index file is no longer updated to match the data set file.

DEACTIVATE *index* [*xxx*_]*database-name* [***[group - name]***]

[AT *node-name*]

index

Description *Required*. Specifies the index file for which logging is to be deactivated.

Format *dset*IX*yy*

where: *dset* = 4-character data set name

IX = entered as shown

yy = 2-character index name

Consideration Before reactivating the index, run the check option of the index maintenance utility program, CSTUIDX, to check the index records and update them where necessary. Check automatically activates the index.

[*xxx*_]*database-name*

Description *Required*. Identifies the database for which index logging is to take effect.

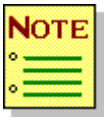
Format

*xxx*_	1–3 alphanumeric characters followed by "_" (optional database prefix)
database-name	6 alphanumeric characters

Consideration If the database is prefixed, the 6-character database name will be preceded by the 1- to 3-character prefix name and an underscore. If no prefix is used, the database name will always be 6 characters.

[***[group - name]***]

Description *Optional*. Identifies the group number if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters

AT *node-name*

Description *Optional*. Passes the command to the remote PDM running at the specified node.

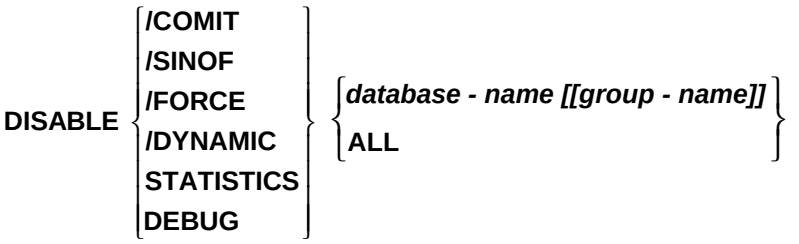
Format 1–6 alphanumeric characters

Considerations

- If you omit this parameter, the command is directed to the local PDM.
- Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Disabling a database (DISABLE)

The DISABLE command signs off each task using the database according to the parameter, and then it unloads the specified database. Once unloaded, you cannot reload the database until you cancel the DISABLE command by entering the ENABLE command (see “Enabling a database (ENABLE)” on page 135). (See also the UNLOAD command in “Unloading a database (UNLOAD)” on page 144.) You can also use the STATISTICS parameter of disable to disable the printing of statistics into the log file, if the database is enabled and loaded.



/COMIT
/SINOF
/FORCE
/DYNAMIC
STATISTICS
DEBUG

Description *Required*. Specifies the object(s) of the DISABLE command.

Options

/COMIT	Signs off each task after its next COMIT or RESET.
/SINOF	Waits until the last task has signed off.
/FORCE	Resets any uncommitted updates and signs off each task immediately.
/DYNAMIC	Dynamically signs off all tasks; unloads and disables the database.
STATISTICS	Disables the printing of statistics into the log file.
DEBUG	Disables the printing of function and status messages at the beginning and end of each function.

Warning: Use of the DISABLE/DYNAMIC operator command overrides the dynslock setting (if you issue DISABLE/DYNAMIC for the database, then the files will be unlocked regardless of the dynslock setting, and the database will be disabled).

Considerations

- Use the ENABLE command to cancel these restrictions.
- The DISABLE/DYNAMIC command is designed particularly for CONTROL:Manufacturing sites. If a task gets dynamically signed off from a database for which you have entered DISABLE/DYNAMIC, then all other tasks running on that database will be forced to sign off, and the database will be disabled.
- The DISABLE/DYNAMIC command overrides the file-locking option provided by the DYNSLOCK PDM input parameter.
- Use the DISABLE/DYNAMIC command when that database is likely to contain logical data corruption. For example, a task which has updated a data set fails to sign off normally when there is no active task log.

{*database - name* | ALL}

Description *Required*. Specifies a particular database or all databases.

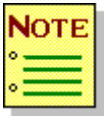
Format 6-, 8-, 9-, or 10- character database name or ALL

Considerations

- The ALL option dynamically signs off each task and disables and unloads all databases.
- The ALL option is not valid with /DYNAMIC.
- Under /DYNAMIC, you must code a database name.

[*[group-name]*]

Description *Optional*. Specifies a group if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

AT *node-name*

Restriction Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Description *Optional*. Directs the command to the PDM running at the specified node.

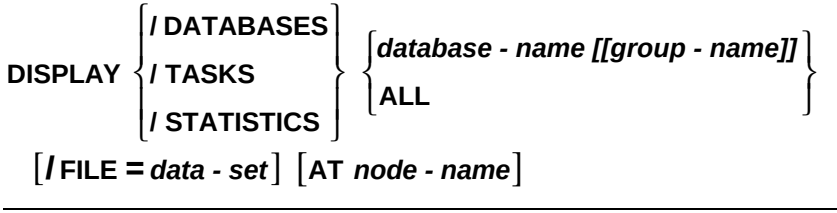
Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Displaying a database (DISPLAY)

The DISPLAY command lists some or all of the following details on the screen according to the parameters you specify:

- The database name
- Whether the database is loaded in systemwide or groupwide global sections
- Group ID and name
- The number of bytes in memory taken up by the database
- The state of the database (Fail, Inactive, or Active)
- The number of active tasks
- The number of active functions
- The number of active threads
- Database statistics



/ DATABASES
/ TASKS
/ STATISTICS

Description *Required*. Specifies the object(s) of the display function.

Options

/DATABASES	Displays the status of one or all loaded databases.
/TASKS	Displays the status of one or all signed-on tasks.
/STATISTICS	Displays statistics for one data set connected to the specified database or all databases to which the specified data set is connected.

Considerations

- Using DISPLAY /STATISTICS may tie up your terminal for some time unless you use the SET OUTPUT (*file-spec*) csiopcom command to direct output to a disk file (see "Communicating with SUPRA Server PDM using csiopcom" on page 148).
- If you are using Multiple Physical Databases, you can display statistics for all databases to which the specified data set is connected by entering:

```
display/statistics all /file=data-set
```

database - name
ALL

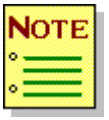
Description *Required*. Specifies a particular database or all databases.

Format 6-, 8-, 9- or 10-character alphanumeric database name or ALL

Consideration The ALL option dynamically signs off each task and disables and unloads all databases.

[*[group-name]*]

Description *Optional*. Specifies a group if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

/FILE=*data-set*

Description *Optional*. Specifies the data set for which statistics are requested.

Format A valid UNIX file name

AT *node-name*

Restriction Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1- to 255-character alphanumeric node name

Dumping a database (DUMPSLF)

The DUMPSLF command causes the PDM to dump the contents of system log file components. You must dump the contents of each system log file component before the PDM can reuse it, and before you can run system log recovery.

The PDM usually carries out this procedure automatically. However, after a system failure, one or both log files may contain data that has not yet been dumped. Alternatively, you may wish to dump remaining data in order to have a complete system log history. You may need this log history in order to run the recovery program, (the PDM does not automatically dump the system log file if it is not full).

Use the DUMPSLF command in the following situations:

- To dump one or both system log file components manually before starting system level recovery
- To dump data from a system log that is not full

Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for a description of system log recovery.

DUMPSLF *database-name* [*group-name*] [AT *node-name*]

database-name

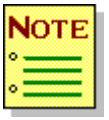
Description *Required*. Specifies the database for which the system log file is to be dumped.

Format 6, 8, 9, or 10 alphanumeric characters

Consideration You cannot dump an active system log file (e.g. if after-image records are still being logged to it). Use the UNLOAD command (see “Unloading a database (UNLOAD)” on page 144) or the DISABLE command (see “Disabling a database (DISABLE)” on page 127) to sign off these tasks.

[*[group-name]*]

Description *Optional*. Specifies the group name if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

AT *node-name*

Restriction Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Enabling a database (ENABLE)

The ENABLE command cancels the restriction imposed by the DISABLE command (see “Disabling a database (DISABLE)” on page 127), allowing tasks to reload a previously disabled database.

ENABLE { ***database- name*** | **ALL** | **STATISTICS** | **DEBUG** } [**[** ***group - name*** **]**] [**AT** ***node - name***]

{ ***database- name*** | **ALL** | **STATISTICS** | **DEBUG** }

Description *Required*. Specifies the object of the ENABLE command.

Format 6, 8, 9, or 10 alphanumeric characters

Options

database-name	Cancels the restriction imposed by the DISABLE command on the specified database.
ALL	Cancels the restriction imposed by the DISABLE command on all databases.
STATISTICS	Sends output to the PDM log file and to OPCOM DML (csiopcom). The only place that does not receive output is the operator device.
DEBUG	Starts printing function and status messages to the PDM log file for each PDML function received.

[**[** ***group - name*** **]**]

Description *Optional*. Specifies a group, if more than one database of the same name is loaded in different groups.

You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

[AT *node - name*]

Restriction Use this parameter only to communicate with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Description *Optional*. Directs the command to the PDM running at the specified node. Omit the AT keyword to pass the command to the local PDM.

Format 1–255 alphanumeric characters

Populating an index (POPULATE)

The POPULATE command populates and activates one index file, which is defined in a currently loaded database, by reading records from the data file and writing corresponding index records to the index file. Once you have run POPULATE, the index is ready for use.

POPULATE *index database-name* [*[group-name]*]

[AT *node-name*]

index

Description *Required*. Specifies the data set to write index records in.

Format *dset*IX*yy*

where: *dset* = 4-character data set name

IX = entered as shown

yy = 2-character index name

database-name

Description *Required*. Specifies the database in which the index is defined.

Format 6, 8, 9, or 10 alphanumeric characters

Consideration If the database is prefixed, the 6-character database name will be preceded by the 1- to 3-character prefix name and an underscore. If no prefix is used, the database name will always be 6 characters.

[*[group-name]*]

Description *Optional*. Specifies the group name, if more than one database of the same name is loaded in different groups.

You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

AT *node-name*

Description *Optional*. Directs the command to the PDM running at the specified node.

Format 1–6 alphanumeric characters

Considerations

- If you omit this parameter, the command will be directed to the local PDM.
- Use this parameter only if you are communicating with a remote PDM through the csiopcom interface (see "Communicating with SUPRA Server PDM using csiopcom" on page 148).

General considerations for the POPULATE command

- The POPULATE command always formats a new index file before writing the index records. If you defined a shadow index, POPULATE copies the completed main index to the shadow index file.
- If population of the main index file fails, both the main and the shadow index files are marked as invalid.
- The POPULATE command locks the data set associated with the index for the duration of the populate operation. It attempts to obtain control of the data set based on the values of RETRIES and INTERVAL (see "Initiating the SUPRA Server Physical Data Manager" on page 81). If it cannot obtain control of the data set, the populate operation is abandoned. Any attempt to access the data set during the populate operation will result in a HELD status being returned to the application after the specified number of retries.
- The populate operation writes messages to the CSIPDMLOG file as the operation proceeds. These messages are the same as those displayed interactively by the Index Utility program described in the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260.

Printing PDM memory (PRINT)

The PRINT command produces a file that is defined by the logical name CSI_DMPANL. This file contains a dump of PDM memory that is used by the csidmpanl utility to produce a readable ASCII-text report. For information on csidmpanl, see “csidmpanl” on page 247.

PRINT *database-name* [*[group-name]*]

database-name

Description *Required*. Specifies the database for which memory is to be dumped.

Options 6, 8, 9, or 10 alphanumeric characters

Consideration If the database is prefixed, the 6-character database name will be preceded by the 1- to 3-character prefix name, and an underscore. If no prefix is used, the database name will always be 6 characters.

[*[group-name]*]

Description *Optional*. Specifies the group name if more than one database of the same name is loaded in different groups.

You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

Setting a database to READONLY access (READONLY)

The READONLY command sets the specified database or all databases to READONLY access, dynamically signs off all active update tasks in the database(s), and unloads the specified database(s). (See "Unloading a database (UNLOAD)" on page 144, and the "Setting a database to UPDATE access (UPDATE)" on page 146.)

READONLY { **/ COMIT** | **/ SINOF** | **/ FORCE** } { ***database - name [[group - name]]*** | **ALL** }

[AT *node-name*]

{ **/ COMIT** | **/ SINOF** | **/ FORCE** }

Description *Required*. Specifies how tasks are to be signed off.

Options

/COMIT	Signs off each task after its next COMIT or RESET.
/SINOF	Waits until the last task has signed off.
/FORCE	Resets any uncommitted updates and signs off each task immediately.

{*database - name* | ALL}

Description *Required*. Specifies whether one or all databases are to be set to READONLY access.

Format 6-, 8-, 9- or 10-character alphanumeric database name or ALL

Options

database-name	Specifies the database to be set to READONLY access.
ALL	Specifies that all current databases be set to READONLY mode, as well as any databases loaded later.

Considerations

- Use the UPDATE command to cancel the READONLY restriction.
- If you use all with the READONLY command, you must also specify the ALL parameter with the UPDATE command. SUPRA Server PDM will not allow you to set all databases to READONLY access and then cancel the restriction for a specified database.

[*[group-name]*]

Description *Optional*. Specifies a group if more than one database of the same name is loaded in different groups.

You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

AT *node-name*

Restriction Use this parameter only if you are communicating with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

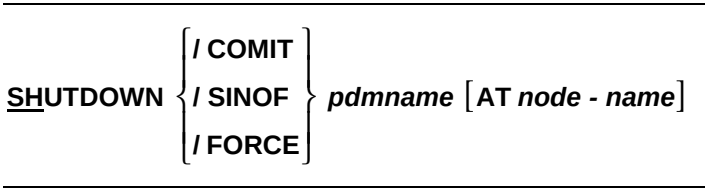
Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Shutting down a database (SHUTDOWN)

The SHUTDOWN command signs off all tasks according to the parameter you specify, unloads all loaded databases, and then terminates the PDM process. Any connected tasks get an ENDT status. SHUTDOWN implicitly disables automatic restart. However, automatic startup will still occur if a new task attempts to access the PDM. You can start up the PDM manually if you wish. See "Initiating the SUPRA Server Physical Data Manager" on page 81 for a description of PDM initiation procedures.



{ / COMIT / SINOF / FORCE }

Description *Required*. Specifies how tasks are to be signed off.

Options

/COMIT	Signs off each task after its next COMIT or RESET.
/SINOF	Waits until the last task has signed off.
/FORCE	Resets any uncommitted updates and signs off each task immediately.

Consideration Any new task attempting a database sign-on will reinitialize the PDM, if automatic PDM initiation is enabled with the logical name CSI_AUTOSTART. Automatic PDM initiation requires the script named in the logical CSI_*nnnnnn* if starting from a remote node.

pdmname

Description Defined by the logical CSIPDMID or CSI_SYSPDMID for multi-systemwide PDMs.

AT *node-name*

Restriction Use this parameter only to communicate with a remote PDM through the csiopcom interface (see “Communicating with SUPRA Server PDM using csiopcom” on page 148.)

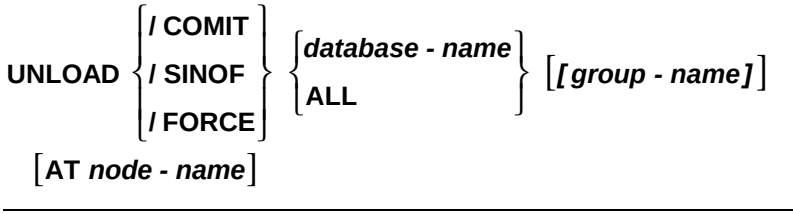
Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Unloading a database (UNLOAD)

The UNLOAD command unloads the specified database or all databases after first signing off each task using it. The database can be reloaded by any subsequent task attempting a sign-on. If you wish to unload the database and prevent any new tasks from reloading it, use the DISABLE command (see “Disabling a database (DISABLE)” on page 127).



{ / COMIT / SINOF / FORCE }

Description *Required*. Specifies how tasks are to be signed off.

Options

/COMIT	Signs off each task after its next COMIT or RESET.
/SINOF	Waits until the last task has signed off.
/FORCE	Resets any uncommitted updates and signs off each task immediately.

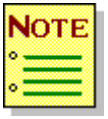
{ *database - name* / ALL }

Description *Required*. Specifies whether to unload one or all databases.

Format 6-, 8-, 9- or 10-character alphanumeric database name or ALL

[*[group-name]*]

Description *Optional*. Specifies a group if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

[AT *node-name*]

Restriction Use this parameter only to communicate with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

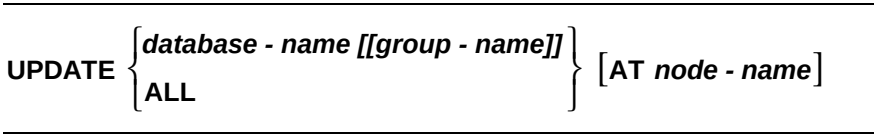
Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Setting a database to UPDATE access (UPDATE)

The UPDATE command cancels the READONLY command, setting all specified databases to allow UPDATE access.



{***database - name***
ALL}

Description *Required*. Specifies whether one or all databases are to be set to UPDATE access.

Format 6-, 8-, 9-, or 10-character alphanumeric database name or ALL

Considerations

- The UPDATE command does not load databases unloaded by the READONLY command.
- If you set all databases to READONLY mode using READONLY ALL, you must use UPDATE ALL to cancel this restriction. If you set a specified database to READONLY mode, you must explicitly specify that database with the UPDATE command to cancel the restriction.

  For example, after the following sequence of commands through the csiopcom utility, the database TESTDB remains in READONLY mode:

  ```
  ==>readonly/comit testdb[125]
  ==>readonly/force all
  ==>update all
  ```

  After you issue the UPDATE ALL, the PDM displays the following message:

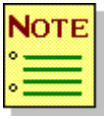
  ```
  CSTI156I  All databases are now available for update
  ```

  However, the database TESTDB remains in READONLY mode because the UPDATE ALL command affects only those databases unloaded by the READONLY ALL command. TESTDB, having already been set to READONLY, is not affected by READONLY ALL or UPDATE ALL. To cancel the READONLY restriction on TESTDB, specify the database name with the UPDATE command as follows:

  ```
  UPDATE TESTDB[125]
  ```

[*[group-name]*]

Description *Optional*. Specifies a group if more than one database of the same name is loaded in different groups.



You must enclose the group name in one set of brackets. The outer set of brackets indicates this item is optional.

Format 1–255 alphanumeric characters enclosed in brackets

AT *node-name*

Restriction Use this parameter only to communicate with a remote PDM through the csiopcom utility (see “Communicating with SUPRA Server PDM using csiopcom” on page 148).

Description *Optional*. Directs the command to the PDM running at the specified node.

Default If you omit this parameter, the command will be directed to the local PDM.

Format 1–255 alphanumeric characters

Communicating with SUPRA Server PDM using csiopcom

csiopcom is a screen-based user interface through which you can enter PDM operator commands. In addition to context-related, online Help, a pop-up menu from which to select PDM operator commands with a single keystroke, and function key support, csiopcom offers two csiopcom commands, LIST and SET.

csiopcom allows you to communicate with any PDM on the network. Therefore, you must always specify the AT *node name* parameter to the PDM operator command to access a remote PDM.

No special privileges are needed to use csiopcom. However, you should ensure that each user has access to a user authorization file through the logical name CSIOPCOM_AUTH. You create user authorization files using the csioauth program (see “Restricting usage of PDM commands using the csioauth program” on page 156). Without a user authorization file, csiopcom users with root privilege can access all PDM operator commands. csiopcom users without these privileges may access only the DISPLAY command.

To run csiopcom and issue SUPRA Server PDM operator commands, type csiopcom. The initial screen displays the Cincom copyright notice.

```
                        SUPRA PDM Operator Interface 1.0
 ____________________________________________________________________________




            (c) Cincom Systems, Inc. 1992
            Use of this software is governed  by a  license
            agreement. This software contains confidential
            and proprietary  information of Cincom Systems,
            Inc. which  is  protected  by  copyright, trade
            secret, and trademark law.




==> DISPLAY/DATABASES
<F1>=Refresh <F2>=Help <F3>=List Cmds <CTRL/D>=Exit
```

You can enter any authorized PDM operator command on the command line at the bottom of the screen. Enter PDM commands in the following format:

PDM command /qualifier parameter [**at *node - name***]

For example:

```
shutdown /comit testpdm at UNIX2
```

The csiopcom interface supports the following function keys:

- **PF1**. Refreshes the screen display
- **PF2**. Displays context-related, online Help
- **PF3**. Lists the PDM commands you are authorized to use in a pop-up menu

Press CTRL-D or type Quit at the command line, to exit to shell.

Refreshing the screen. Press function key PF1 to refresh the screen and delete any characters from the command line.

Displaying online Help. If you have defined the logical name SUPRA_HELP, you can press function key PF2 at any stage during csiopcom processing to display context-related help. For example, if you type display at the command line and then press function key PF2, the following Help screen displays:

```
csiopcom

DISPLAY

Displays the current status of specified databases or tasks,
listing the following details on the screen according to the
parameters and qualifiers you specify:

     o  Database name (6, 8, 9 or 10 characters)
     o  If the database is loaded in system-wide (S) or group-wide
        (G) global sections
     o  group id and name
     o  The number of bytes in memory taken up by the database
     o  The state of the database, Fail, Inactive or Active
     o  The number of active tasks
     o  The number of active functions
     o  The number of active threads
     o  PDM statistics

Press RETURN to continue ...
```

The online Help is useful if you are unsure of the syntax of a PDM command.

Displaying a pop-up menu that lists PDM commands. The DBA can use the csioauth command to restrict the PDM commands available to certain users. Press function key PF3 to display the PDM commands (that you have been authorized to use) in a pop-up menu.

```
                        SUPRA PDM Operator Interface 1.0
_____________________________________________________________________________

    Valid SUPRA PDM
    commands are :

    A - DISPLAY
    B - UNLOAD
    C - SHUTDOWN
    D - READONLY
    E - UPDATE
    F - DISABLE
    G - ENABLE
    H - DUMPSLF
    I – PRINT
    J – ACTIVATE
    K – DEACTIVATE
    L – POPULATE

    Select option letter or
    press RETURN to exit :

==>
<F1>=Refresh <F2>=Help <F3>=List Cmds <CTRL/D>=Exit
```

The above example shows access to all the PDM operator commands. Type the key letter (A through L) to select a PDM command. Do not press RETURN. csiopcom displays the command keyword (DISPLAY, UNLOAD, SHUTDOWN, etc.) at the command line ready for you to type any qualifiers and parameters. When you press RETURN after entering a PDM command, the screen clears, and the output from the command is displayed at the top left of the screen.

Using csiopcom commands: LIST, REDO, SET and QUIT. csiopcom supports four commands, LIST, REDO, SET, and QUIT. These commands are not for communication with the PDM, although their syntax is similar to that of PDM commands.

LIST

Description Displays all authorized SUPRA Server PDM commands in a window from which you can select a command.

Considerations

- Do not use this command when running in batch.
- This command works only from the csiopcom utility. It is equivalent to pressing function key PF3.

REDO

Description Allows the previous command to be re-executed without retyping it.

Consideration May be abbreviated by simply typing r or R.

The SET command sends input to and output from csiopcom to both the terminal and a disk file, or to a disk file only.

SET { **LOG (*file - name*)** | **NOLOG** | **OUTPUT (*file - name*)** | **NOOUTPUT** }

{ **LOG (*file - name*)** | **NOLOG** | **OUTPUT (*file - name*)** | **NOOUTPUT** }

Description *Required*. Specifies the disposition of csiopcom logging.

Format Valid UNIX file specification

Options

LOG (*file-name*)	Logs all input and output to the specified file, and displays it on the terminal screen.
NOLOG	Terminates logging.
OUTPUT (*file-name*)	Sends all input and output to the specified file.
NOOUTPUT	Sends output to the screen.

Considerations

- You can enter the SET command only from the csiopcom utility to the PDM.
- The SET OUTPUT (*file-name*) command is particularly useful with the DISPLAY STATISTICS PDM operator command. It allows you to send the statistics to a file without scrolling through the entire display on your terminal.

QUIT

Description Exits CSIOPCOM and returns to the shell.

Consideration Equivalent to CTRL-D.

Writing your own interface to SUPRA Server PDM

You can write your own user interface to the PDM using the OPCOM DML command. Refer to the *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240, and the Message Relay Facility described below.

SUPRA Server PDM messages are output by the central PDM image (csipdm), by csidatbas, and by the System Log Dump Program (cstudslf). To direct these messages to a named pipe that you can then read from your program, you must define the following items:

- The logical name, CSI_MRELAY, that causes all messages from csidatbas to go to a named pipe.
- The number of messages SUPRA Server PDM can send to the named pipe is determined by the maximum size of a named pipe on your system (the name of the system parameter varies). The following may be reasons that the PDM cannot send any more messages to the named pipe:
  - There is no active named-pipe-reading program.
  - The program is not picking up messages fast enough.
- If the PDM cannot send any more messages to the named pipe, it will keep a count of the number of messages that it was unable to send. On the next successful send, it will output the following message:

  MESSAGES LOST = *nnnn*.

If the input file parameter MRELAY is Y (that causes all console messages from the PDM to go to a named pipe), and the logical name CSI_PDMID points to TESTPDM, and the PDM is systemwide, then the named pipe will be TESTPDM_000000. Alternatively, if the PDM is running in the group 145 with the logical name CSI_PDMID pointing to QADBD, the named pipe will be QADBD_000145.

See "Initiating the SUPRA Server Physical Data Manager" on page 81 for a full description of the PDM input file parameters.

Example named-pipe-reading program. The following is a skeleton message reading program in pseudocode. You should start your reading program before the SUPRA Server PDM (csidatbas and/or csipdm) starts sending messages and fills up the pipe. See “Example mailbox read program” on page 263 for an example of a mailbox read program written in C programming language.

```
Program Pipe_read
Begin code
      Construct pipe name using the translated value of CSI_PDMID
  and its    group number (000000 if System wide)
      Create the pipe
      While (ok to continue)
      begin-while
              read from pipe
              do action on message received
              decide if ok to continue
      End-while
End code
```

Restricting usage of PDM commands using the csioauth program

SUPRA Server allows you to create authorization files containing subsets of PDM commands that you can then associate with one or more PDM users. You include the PDM commands in the authorization file by running the user authorization program csioauth and replying to the prompts. Then you define the logical name CSIOPCOM_AUTH to point to this file.

By default, users with root privileges have access to all PDM operator commands through csiopcom, if there is no logical definition for CSIOPCOM_AUTH. Users without root privileges can access only the DISPLAY operator command unless they have a user authorization file that gives access to other PDM commands.

Create a user authorization file and make it available to a specified user or group of users in two stages:

1. Run csioauth and reply to the prompts to create an authorization file.
2. Define the logical name CSIOPCOM_AUTH pointing to the authorization file you created. If you define CSIOPCOM_AUTH at group level, all users in the same group have access to the same PDM commands. Alternatively, if you define CSIOPCOM_AUTH at process-level, users in the same group can access different PDM commands.

A process-level logical definition takes precedence over a group-level definition.

To run the authorization file creation program csioauth, you need to have DBAPRV privileges. The following sequence of screens illustrates how to create an authorization file. Run the authorization program by entering the following command:

```
csioauth
```

This displays the authorization screen for the first PDM command, DISPLAY, as follows:

```
SUPRA PDM Operator Command  Authorization Program n.n
 ____________________________________________________________________________


Permit SUPRA PDM Command <DISPLAY    > (Y/N/<CTRL>/D) :

            (c) Cincom Systems, Inc. 1992
            Use of this software is governed  by a  license
            agreement. This software contains confidential
            and proprietary  information of Cincom Systems,
            Inc. which  is  protected  by  copyright, trade
            secret, and trademark law.
```

Valid responses include:

- Y to allow a PDM command
- N to deny a PDM command
- CTRL-D to skip to the prompt for the authorization file name, or if the authorization file name prompt is displayed, to abandon the selection

CTRL-D causes csioauth to deny all subsequent PDM commands. Thus, if you press CTRL-D after having permitted the first two PDM commands, all other PDM commands will be denied. If you press CTRL-D at the first prompt (authorization for the DISPLAY command), the authorization file will allow only the DISPLAY command.

As you reply to the “Permit SUPRA PDM Command” prompt, including and excluding PDM commands from the authorization file, the results are displayed on the screen as follows:

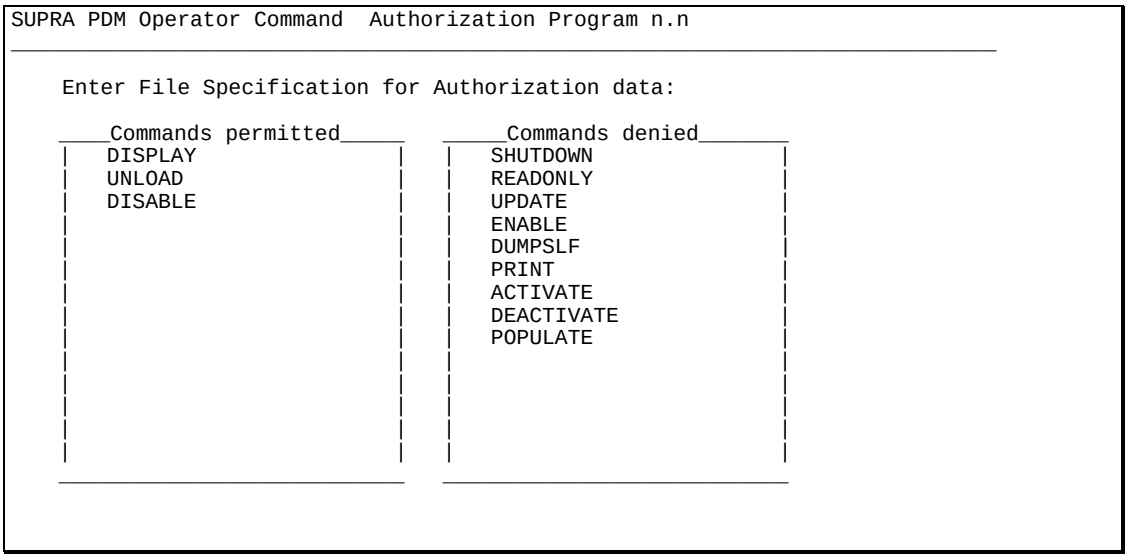
```
SUPRA PDM Operator Command  Authorization Program n.n
_____________________________________________________________________________

    Permit SUPRA PDM Command <PRINT   > (Y/N/<CTRL>/D)

    ____Commands permitted_____    _____Commands denied_______
    |   DISPLAY                |   |   SHUTDOWN               |
    |   UNLOAD                 |   |   READONLY               |
    |                          |   |   UPDATE                 |
    |                          |   |   DISABLE                |
    |                          |   |   ENABLE                 |
    |                          |   |   DUMPSLF                |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    |                          |   |                          |
    _____________________________    ____________________________
```

The above screen shows a partially completed authorization session. It is prompting the user to define the authorization for the PRINT command, listing the PDM commands permitted or denied so far.

Once you have permitted or denied the last SUPRA PDM command, csioauth prompts you to enter the file specification for the authorization file as follows:



Once you have successfully created an authorization file, define the process-level logical name CSIOPCOM_AUTH to point to it as follows:

```
csideflog CSIOPCOM_AUTH /home/user/mike/opcom.auth
```

Communicating with the PDM using the csireply command

The csireply command is used to nominate operator terminals and to reply to requests sent to operator terminals. To use this command, the user must have the REPLY privilege. See “Maintaining user privileges (csichkpriv)” on page 56 for procedures to define user privileges. Also see “Enabling communication between processes and nominated operators (csioper)” on page 65 for information about communications between processes and nominated operators.

Messages sent to operator terminals are time-stamped, and they also contain the process ID of the process sending the message. The operator for which it was intended is also sent to cater for terminals that have been assigned to multiple operators; for example:

```
%%%%%% Wed Jul   1 15:15:29 1992  %%%%%%
Message for OPER3 from process 1234
CSI001I  alecs,  GROUP wide SUPRA PDM  release  1.0
active on node mickey
```

Messages are only sent to the operator terminal once.

Setting up the csireply command

Take the following steps to use the csireply command:

1. Set the PDM input parameter SYSOPCOM equal to Y, the default. At regular intervals, the PDM will display the message “CSI0006O Reply with a SUPRA PDM command” at enabled PDM operator terminals.

2. Set the PDM input parameter OPERATOR equal to OPER*n*, where *n* is the number of the operator you wish to be the PDM operator. This should be a number in the range 1–12, with 1 being the default value.

3. Enable one or more operator terminals using the csireply command (see “Using the csireply command” on page 163); for example:

```
$ csireply -e oper3
```

The PDM sends all messages that do not have a severity level of L (log file only) to the nominated operator console unless stated otherwise by the CONSOLE PDM input parameter. You can specify only one operator per PDM, but you can enable as many operator terminals per PDM as you wish.

It may be advisable to have at least one terminal on your site dedicated for use as a PDM operator terminal. This terminal can be used to enable constant monitoring of one or more PDMs and allow corrective action to be taken immediately. Using this procedure prevents the need to log into different accounts to issue commands to different PDMs.

Requests sent to operator terminals are similar to messages in that they are time-stamped and contain the process ID of the sending process. Each request is also given a unique identifier that also forms part of the request sent to the operator terminal(s); for example:

```
%%%%%% Wed Jul   8 12:32:29 1992
Request 123 for process 1234
CSI0006O Reply with a SUPRAPDM command
```

Unlike messages, the text of a request is repeatedly sent to the operator terminal(s) until the message is acknowledged. The interval between the request being sent is controlled through the logical name CSI_REPLYTIMER. This interval is specified in the number of minute intervals. All requests are sent simultaneously at this interval, regardless of where in the interval they are initiated. Response to a request is given through the csireply command specifying the request for which it is intended.

If a terminal has been enabled for multiple operators, or if two PDMs share the same operator, you will receive messages and prompts for all these PDMs. In this instance, you will have to take extra care not to accidentally send a command to the wrong PDM.

The database access module (csidatbas.o) also outputs messages to an operator terminal. The operator to whom these messages are sent is determined through the logical name CSI_CONSOLE, which should evaluate the operator to be used; for example:

```
csideflog -g CSI_CONSOLE OPER1
```

If the logical name is not defined, messages are not sent to an operator terminal.

The PDM issues prompts at operator terminals at regular intervals. The message displayed will appear as follows:

```
%%%%%%  Wed Jul  8 12:56:27 1992 %%%%%%
Request 123 for process 4321
CSI0006O ALECS, Reply with a SUPRAPDM command
```

Reply to this prompt using the csireply command (see “Using the csireply command” on page 163); for example:

```
csireply -t 123 "display/databases"
```

The above example will display all databases loaded in the PDM ALECS.

Using the csireply command

The format of the command is:

csireply { **-e OPER*nn*** | **-d OPER*nn*** | **-s** | **-t *request* [" *message*"]** }

{ **-e OPER*nn*** | **-d OPER*nn*** | **-s** | **-t *request* [" *message*"]** }

Description *Required*. Specifies terminal use, status or a response to a request to disable.

Format

OPER*nn*	1–12 or all
request	Decimal number
"*message*"	1-*n* alphanumeric characters enclosed in quotes

Options

-e OPER*nn*	Specifies the operator to be enabled. When used, a list of operators for the terminal is displayed.
-d OPER*nn*	Specifies the operator to be disabled. When used, a list of operators for the specified terminal is displayed. If the operator to be disabled is not enabled for the terminal, an error is displayed.
-s	Displays the operator status of the current terminal. If no operators are currently enabled for the terminal issuing the command, a message indicating the condition is displayed.
-t *request*	Sends a reply to a request. The request is specified as a decimal number.
"*message*"	Optional message sent to an operator request.

Considerations

- When you use the -t *request* option, the request specified must be valid; otherwise, an error message is displayed.
- If you use the operator option, all operators can be enabled or disabled in one operation using the keyword ALL.
- When you use the message option, the message request is canceled once it has been replied to.

Examples

- The following example enables operator 1 on the current terminal:

```
csireply -e oper1
```

- The following example disables all operators on the current terminal:

```
csireply -d all
```

- The following example displays operator status of the current terminal:

```
csireply -s
```

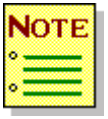
- The following example sends a message to shutdown PDM ALECS and, therefore, terminates request number 123:

```
csireply -t 123 "SHUTDOWN/FORCE ALECS"
```

6

Tuning your database

SUPRA Server is designed to provide optimum performance. If installed and implemented correctly, it provides good response time and efficient work throughput. If performance deteriorates (response time increases, and batch jobs and application programs execute slowly), you need to examine the way your system is running and look for ways to improve performance.



This chapter contains suggestions to help you tune your SUPRA Server system to improve overall performance. It is important to use the methods given in this chapter as guidelines only, not rules. You will find that some methods have more effect on performance than others; tuning is often a question of trial and error.

Tuning can be broken down into the following general areas:

- System tuning
- Physical database tuning
- Logical database tuning
- Optimizing program design

This chapter discusses system tuning, physical and logical database tuning, and efficient program design. See “Initiating the SUPRA Server Physical Data Manager” on page 81 for detailed discussions of parameters that you can use to tune your database.

System tuning

The following UNIX file system parameters affect the performance characteristics of large data sets:

Parameter	Description	Recommended value
bsize	Block size	Equals the block size for all PDM data sets stored in the file system. For example, suppose your database contains some large data sets that have a 4096 block size (records per block * logical record size rounded up to the nearest 512). Cincom recommends that these data sets reside on a file system defined with bsize = 4096.
fsize	Fragment size	Equals the bsize parameter.
nbpi	Number of bytes per node	Equals a large number in order to reduce the size of the node space reserved in the file system. In theory, this number could be the size of the smallest file on the file system.
maxbpg	Maximum blocks per cylinder group	One hundred percent of the size of the cylinder group as determined by the dumpfs command described later. This setting causes large files to fill a cylinder group before starting to allocate space in the next cylinder, keeps the files more compact on the disk, and reduces seek time. The default setting is 1/4 the size of a cylinder group. This setting spreads large files out into multiple cylinder groups, thus, increasing the seek time.
rotdelay	Rotational delay	Equals zero for most systems.
contiguous	Makes the file system contiguous	The contiguous parameter is an option when creating a file system in the Logical Volume System (LVS). This parameter will help keep the blocks of large files as close together as possible on the disk.

You must set the *bsize*, *fsize*, *nbpi*, and *contiguous* parameters when the file system is created.

The *maxbpg* and *rotdelay* are tunable parameters that you can modify after the file system is created. Use the following command:

```
tunefs -d rotdelay -e maxbpg  filesystem
```

Unmount the file system before executing the tunefs command. You can determine the cylinder group size and all other file system parameters using the following command:

```
dumpfs filesystem | more
```

Tuning your physical database

The first step in designing an efficient database is to normalize your organization's data. Data normalization involves reducing the transactions and data flow pathways to their simplest form. You can then use this framework as the basis for the physical database design. The *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, explains how to normalize your data.

Accessing files on a network

If you are using NFS mounted file systems, it is possible to have database files residing on machines remote from the one on which the PDM is running. Data integrity cannot be guaranteed because no method can assure that on return from an I/O request, data has been physically written to disk.

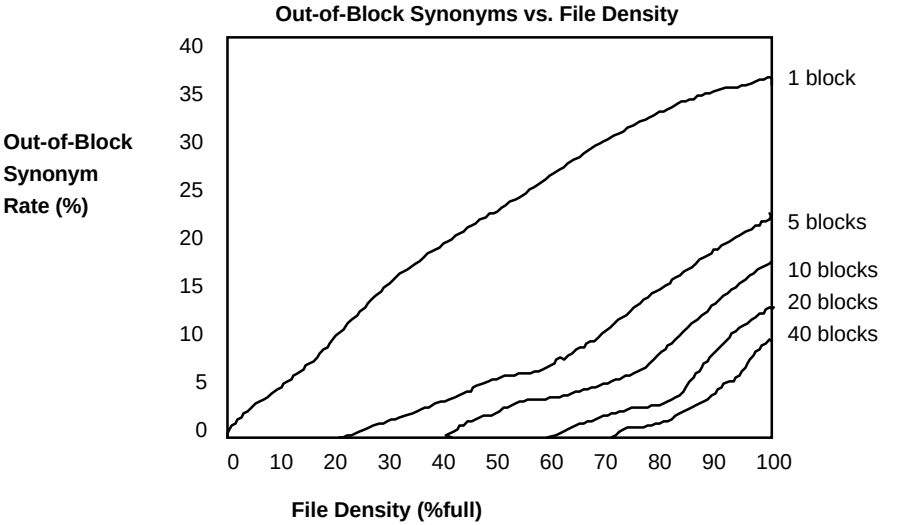
Optimizing primary record retrieval

The PDM uses a hashed addressing technique, based on the value of the record key, to calculate a home address on the disk for each primary record. The record key is randomized to give a relative record number (RRN) or home address. In some cases, two or more record keys randomize to the same RRN. These records are known as "synonyms." When synonyms occur, SUPRA Server places the first record in the home address and chains all subsequent synonyms together as shown below:

A, B, and C are primary records whose record keys have all randomized to the same RRN. To retrieve record C (the last record in the chain), the PDM calculates the home address, retrieves A, finds that the record keys do not match, follows the chain to B, finds that the records keys still do not match, and then follows the chain to retrieve C.

The PDM attempts to place all synonyms in the same physical block so programs can retrieve any record in the chain with only one physical I/O. The physical block in which a record ought to be found is known as the home block for that record. If there is not enough room in the home block for the synonym, the PDM places it in another block. Each block that the PDM has to search to retrieve a given record entails an extra physical I/O. The worst case, using the above example, would be if each synonym were in a different block. Three physical I/Os would then be necessary to retrieve record C.

To optimize system performance, you should try to minimize the number of out-of-block synonyms. Because synonyms are bound to occur, you need to leave a certain percentage of file space free so most records can be placed in the home block. The following figure shows the out-of-block synonym rates measured at various packing densities and blocking factors.



- **File density.** The number of records in use as a percentage of file capacity; also known as the packing density
- **Blocksize.** The number of records per block

You can substantially reduce the out-of-block synonym rate by lowering file density and increasing block size. As a general rule, a file density of 80% provides an acceptable level of out-of-block synonyms. The ideal block size depends upon your type of application and the space you have available. However, it is possible to increase block size to such an extent that it takes too long to read a list. Therefore, you must balance the benefits of reducing out-of-block synonyms against the data transfer time. A large data transfer time may give your task improved performance; however, it will degrade system efficiency. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for information about setting the blocksize.

Avoiding fragmented files

It is important to keep disk files contiguous. The SUPRA FORMAT function attempts to create contiguous files; however, if this is not possible, FORMAT will create fragmented files and warn you if the file is in more than three fragments. You may also create fragmented files using the unload/reload utilities, in this case, no warning is displayed.

Fragmented files cause a deterioration in performance. To eliminate fragmentation on a SUPRA Server file system disk partition, follow these steps:

1. Disable all databases that use the SUPRA Server file system disk partition.
2. Use a standard UNIX utility, for example, tar, to back up the SUPRA Server file system disk partition.
3. Use csicp to copy all the files from the SUPRA Server file system disk partition to a standard UNIX directory.
4. Use a standard UNIX utility, for example, tar, to back up the files created in Step 3.
5. Reinitialize the SUPRA Server file system disk partition using csimkfs.
6. Use csicp to copy all the files created in Step 3 to the SUPRA Server file system disk partition.
7. Enable all databases that were disabled in Step 1.

Avoiding fragmented chains

The record chains of related data sets that are often updated may also become fragmented. Ideally, all the records for any given chain should start in the same control interval. To avoid the deterioration in performance caused by fragmented chains, run Fast utilities on volatile related data sets regularly. This will bring fragmented chains back into the same control interval. Refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220, for procedures on using Fast utilities.

Using primary data sets

You should retain primary data sets for key access only. Nonkey data can be moved from primary data sets to related data sets containing record codes. (See “Using coded records” on page 174 for a description of the advantages of using record codes.) This reduces the size of primary records, allowing more records-per-block and fewer out-of-block synonyms. Primary data sets containing transaction data can be converted to related data sets. For instance, if a primary data set contains a large amount of optional comment data, you can place this comment data in an associated related data set.

Another important consideration for primary data set usage is the amount of wasted space in each block. For example, if records are 260 bytes long and blocks are 512 bytes long, each block holds one record plus 252 bytes of wasted space. You can avoid this wasted space by reducing the size of each record by 4 bytes, if possible, or by increasing the block size so that a smaller proportion of the file is wasted. This is a major consideration when you use very large files.

Using related data sets

The PDM stores related records in linked lists. Each related record may be associated with one or more primary records. The objective of blocking related data sets is to maintain a linked list of average length within one block. This means only one physical I/O is needed to perform successive actions on that list. You specify the block size in the LOGICAL-RECORDS-PER-BLOCK field when you define the file specifications for related data sets. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information on specifying block size.

To calculate the optimum block size for your application, establish the average length of a list of related records connected by a primary linkpath. You may need to return to the file specification screen several times to alter and test the effect of different block sizes. The values you enter for control interval and load limit also affect whether or not a list of records overflows onto another block. The following sections discuss these considerations in more detail.

Defining the control interval and load limit

You define values for control interval and load limit at the file specification screen in DBA. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for information on the DBA specification screen. The control interval specifies the number of related records whose allocation the PDM controls as a single unit. The load limit specifies the load capacity for the control interval as a percentage.

The control interval and load limit values affect the addition of records to a related data set in the following ways:

- If an application program adds a record to an existing list of related records, the PDM places that record in a control interval that contains other records in the same list, if there is any space.
- If the record is the first record of a new list of related records, or belongs to a list in a full control interval, the PDM puts it in a control interval that is below its load limit, if there is one.
- A control interval that is above its load limit accepts records only in lists that have already started in that control interval. When all control intervals are above their load limit, new lists may start in one of these control intervals, and subsequent SINOFs return the LOAD status.
- If you set the LOAD-LIMIT to zero, the first record of a new list is located in the next higher control interval from the last new chain started. This allocation process continues, starting from the first control interval to the last, and then starting over from the first.

Sometimes a new list starts in a block in that other lists have already started, because the block is in a control interval that is below its load limit. This could cause parts of each list to overflow into different blocks or control intervals if many other additions are performed. One technique that avoids this situation when loading a related data set is to force each new list to start in a new control interval by setting a low load limit. You can later use the DBA RESET function to increase the load limit after determining the average length of your lists. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for a description of how to use the RESET function.

The second method of keeping lists from overflowing into different blocks is to set the load limit to zero. Doing so causes all control intervals in the data set to be evenly distributed. Each new list starts in a new control interval until the last control interval is used. Then, it starts over with the first control interval. This allocation process continues until the data set is full. If there is space, records added to existing lists go in a control interval that contains other records of the same list.

Establishing the primary linkpath

A related record can be a member of as many relationships or linkpath chains as needed. However, because only one occurrence of the record exists for all the relationships, it cannot be placed in an optimum physical location for each linkpath of which it is a member. It is important to identify the primary linkpath for each related data set and ensure that as many programs as possible use it when adding data. To determine the primary linkpath, you need the following information:

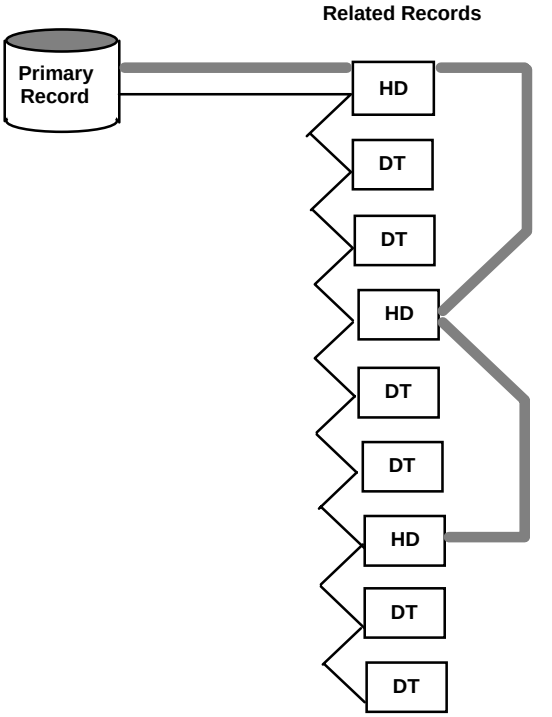
- The average length of each chain
- The frequency of access along each chain

In theory, the primary linkpath should be either the longest chain or the one most frequently accessed. In practice, the primary linkpath is usually a compromise between the two.

To ensure that programs use the primary linkpath, specify the primary linkpath to the PDM through the linkpath parameter.

Using coded records

Processing long lists of related records is time consuming. You can alleviate this problem by using coded records with a header-and-detail structure. For example, in the following figure, the lines on either side of the related records represent different linkpaths. You can scan down the HD records until you are close to the required record. You can then change linkpaths and scan serially through the DT records.



It is more efficient to use code-directed reads than to read all the records on the linkpath until you find a record with the required code.

Evaluating redundant data items

Redundant data items are data items that appear in more than one data set. Redundant data items improve the speed of data retrieval. However, they also increase the processing time needed to update information and can cause database inconsistencies. Before you define redundant data items, consider the trade-off between speed of retrieval and speed of update.

Managing record holding

Record holding is a facility within the PDM that protects records from being updated by two tasks at the same time. A task must reserve (hold) records before it can update them. Because only one task at a time can hold a record, record holding controls contention among tasks accessing the same record. The DML commands that hold records while they execute are: ADD-M, ADDVA, ADDVB, ADDVC, ADDVR, CNTRL, DEL-M, DELVD, RDNXT, READD, READM, READR, and READV. Records may be held differently depending on the processes being performed.

Because more than one task may be accessing the same file in a multitask environment, concurrent updating of a record by two or more tasks could occur, and this could destroy database integrity. To prevent this, the PDM holds any records to be updated by a task. Single-task also performs record holding when the SINON mode is UPDATE.

The PDM holds the record until the record is no longer required (function completion, COMIT processing, and RESET). The DML commands do two types of record holding (numbers 1 and 2 below); the PDM does one type of record holding (number 3 below):

1. **Implicit record holding.** The PDM temporarily holds records to prevent interference between DML commands. The PDM temporarily holds additional records that might be needed for completing a DML function (a primary record on ADDVC). The following commands perform automatic record holding: ADD-M, ADDVA, ADDVB, ADDVC, ADDVR, DEL-M, and DELVD. The PDM holds internally all the records affected by the ADD or DELETE (synonym chain of primary records), not only the record to be added or deleted. The ADD or DELETE may change the linkpaths in the affected records. The system determines which records could be affected and holds all of the affected records for the duration of the DML command.

   Automatic record holding is in operation in both task logging and nontask logging environments. If task logging is active and the held record(s) was updated, the automatically held record(s) become uncommitted held record(s) (see number 3) after the DML function completes. Otherwise, these records are released for use by other tasks when the DML completes.

2. **Explicit record holding.** An explicit hold is required if you want to update a record or if you want to prevent someone else from updating a record. Explicit record holding is accomplished with read commands having END. in the last parameter instead of RLSE. The read commands are: RDNXT, READD, READM, READR, and READV. You can explicitly hold only one record per file per task if the value for the PDM input parameter MULTIHOLD = N. If MULTIHOLD = Y, you may hold as many records as can be retained in the holding table. Your next DML command to that file can then update this record(s).

   You can explicitly hold records in both task logging and nontask logging environments. The only time explicit record holding is not performed is in a multitask environment without task logging when a file is opened for READONLY access and the reads use END. in the last parameter.

   If task logging is not active, a task requesting a record already explicitly held by another task is not placed into a wait state, but immediately receives a HELD status.

   The PDM releases an explicit hold on a record if you:

   - Issue another read command to the same file with END. in the last parameter.
   - Issue a FREEX command for the file.
   - Issue a COMIT or RESET command for the logical unit of work.
   - Issue a WRITE command for the record (WRITM, WRITV, and ADDVR). If task logging is active and the task updates the record, the explicitly held record becomes an uncommitted held record (see number 3).

3. **Uncommitted record holding.** If task logging is active, records previously held, either explicitly or implicitly, become uncommitted records after an update to the record completes successfully. Uncommitted record holding by the PDM prevents interference between transactions that could jeopardize database integrity. Once a task updates a record, it and any associated held records cannot be updated by any other task until the updating task has COMMITTED or RESET.

   Uncommitted record holding is not performed by specific DML commands. The PDM provides uncommitted record holding to allow updates to be backed out or committed. Therefore, these records remain uncommitted (and held) until you issue your next COMIT or RESET command. At that time, the records are released and become available for use by other tasks.

   In some cases, two logical units of work may request the same database records simultaneously. For example, Transaction A holds Record x and starts processing. Meanwhile, Transaction B holds Record y and starts processing. At some point, Transaction A tries to hold Record y. Since y is already held by B, Transaction A receives a HELD status. During Transaction B's processing, it tries to hold Record x, that is held by A. Therefore, if B waits, neither A nor B will ever complete since they are waiting for each other.

   This situation is called a "deadlock," "deadly embrace," or "fatal embrace." The PDM prevents it by returning a HELD status to Transactions A and B, indicating that a record could not be reserved for either task because of a deadlock situation.

   Any task receiving a HELD status should issue a RESET and then retry its processing from the most recent commit point. This RESET command resolves the deadlock by releasing all held resources for the logical unit of work. Contention between logical units of work can cause poor performance. Therefore, we recommend that you design your logical units of work so that different logical units of work in different tasks do not require the same database records.

Recommendations for managing record holding

Cincom recommends the following actions regarding record holding:

- If a process requires exclusive use of your database, you can use the SINGLE mode which bypasses all record-holding logic and disables task level recovery, and thereby improves performance. Recovery from a failure while running in SINGLE SINON mode requires a full database restore. The SINGLE SINON mode is supported by both single-task and multitask SUPRA PDM.

- If you run your database in a multitask environment, the number of records a task may hold is dictated by the size of the holding tables, which in turn is defined by the DBA through the Database Details screen.

- Monitor the size of the holding tables through the Database Details screen. As the number of held records increases, the time taken to search the holding table increases. Note that the holding table is a pool of entries for all users. Thus, if you define 10 users of the database with 100 held records each, one user holding 1000 records could use the entire holding table. Remember that for every related key you delete, update or insert, you could hold more than one record.

Defining logical units of work

SUPRA Server is designed to process logical units of work (transactions). A logical unit of work is a group of database requests that must all be completed together or not at all. A task may consist of one or several logical units of work. For example, a logical unit of work could be entering a purchase order to the system, posting an invoice, or adding a new customer.

In a task logging environment, you define the length of a logical unit of work by issuing COMIT commands. The size of a logical unit of work is from commit point to commit point. The length of a logical unit of work determines how far back (in processing) you need to go to restart after an error. Again, depending on whether task logging is active, you might have to go back to the beginning of your program or just a few steps to the most recent COMIT. In a task logging environment, all updates and relevant information are written to a task log file on a task and a logical-unit-of-work basis. For more information on task logging and the task log file, refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260.

The following example illustrates a sample program structure that could be used to implement a logical unit of work:

```
INITIALIZATION (SINON)
While not finished, do:
   screen input and validation
     Database Update Processing (VMS DML commands)
     If error, then RESET
          else COMIT
end
TERMINATION (SINOF)
```

The initialization (SINON) process identifies the new task to the PDM and allocates a unique internal task identifier. Once you have signed on, you can begin processing.

You can process the database using the Physical View DML. Your program can perform read-only database calls or can update database records. If your program updates database records, the records are held until committed, and they cannot be accessed by other programs during that time.

If an error is detected while updating, you can remove the updates to the records by issuing a RESET. If no errors are detected, issue a COMIT to free any held records. Updates to the records are now permanent; they can no longer be removed by issuing a RESET. Errors in a logical unit of work are handled differently based on where the error was detected:

- **Error detected by the program.** Because the logical unit of work is incomplete when the error is detected, all the processing done before the error must be undone by issuing a RESET. Your program can either start a new logical unit of work or sign off, depending on the severity of the error.

- **Error not detected by the program.** If you code your program to execute COMITs at appropriate times, the PDM can assure the integrity of your database. The PDM ensures that each logical unit of work is always complete. If the PDM finds that a program has terminated abnormally (batch and online) before signing off, the PDM resets to the most recent commit point and signs off the task.

Managing buffers

Buffer organization and buffer use at run time affect performance. There is a trade-off between memory and disk I/O performance. As your buffers become larger and more numerous, they occupy more memory, but your disk I/Os become fewer and larger. As you reduce the number and size of your buffers, you increase the memory available at the cost of disk I/O performance. However, because commercial applications more often suffer from being I/O-bound than memory-bound, you should aim to improve disk I/O.

Consider the following when deciding on your buffer strategy:

- You can allocate a pool of buffers defining the areas used for input from and output to primary and related data sets. The number of buffers in the buffer pool depends on how you use your database. For example, a multiuser database needs a larger pool of buffers than a single-user database.
- The PDM flushes the buffer contents to disk whenever a task performs a COMMIT or a SIGNOFF. The PDM also writes the buffer contents to disk when a task needs a block that is not in memory and no unused buffers exist. A RESET also causes buffer flushes.
- When a task issuing a commit opens data sets for updating, it also updates the corresponding buffers. After updating, the PDM writes all buffers to the task log and the system log before the block is written to the data set.
- The size of a data set record helps determine buffer size. The PDM calculates the size of a buffer automatically using the maximum block size of the data set. For example, if a data set contains five 300-byte records per block, its block size must be at least 1536 bytes.
- It is better to avoid sharing buffers across data sets although it can be useful to share buffer pools among data sets with the same block size.
- Related data sets are organized with all records on a chain as close together as possible. A large value for records-per-block allows many serial accesses without disk I/O. It is helpful to specify a fairly large number of records-per-block to cut down the number of I/Os. The number of I/Os is generally more significant from a performance standpoint than the amount of data transferred in an I/O operation.

- Another factor affecting the number of copies of buffers is simultaneous use of the database by multiple tasks. If the only copy of a buffer is in use (a function required a buffer that is executing), then another task must wait until the buffer is free, perhaps severely degrading performance. In a multitasking environment, you should monitor buffer handling to avoid thrashing. Thrashing takes place when excess I/O prevents useful work. You can change the number of buffer copies (but not the number of records-per-block), alter the way in which buffers are shared between data sets, recompile the database, and monitor its performance.

- You can improve the initial load performance of large primary data sets dramatically by changing the number of copies of the buffer used by the data set to 1, and sorting the input file into ascending RQLOC sequence. These changes make file I/O sequential. Also, the load application should sign on to the database in SINGLE mode and should be linked with the single-task PDM (csibatbas.o).

Improving database performance with buffers

Database performance tuning is a complex, ongoing activity dependent on numerous criteria and requiring regular monitoring and adjustments. There is no single setup that is optimal for all sites or even for the same site at different times. However, one of the most important ingredients in tuning any application's performance is its use of memory buffers. Tuning buffers use will allow you to maximize the benefits of PDM buffers. To effectively tune buffer use, it is helpful to review how buffers are searched.

Understanding buffer search algorithms

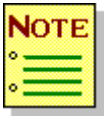
When data records are accessed from a SUPRA Server PDM database, they are maintained in memory buffer pools. These buffer pools are searched when an application task requests a record. The following search algorithms are used:

- **Serial Scan**. The list of buffers is linearly searched from beginning to end until the desired buffer is found. This method is used by the PDM when there are fewer than 20 buffers in a buffer pool.
- **Hashing**. An internal algorithm determines (within 20 buffer searches) the actual buffer location within the list of buffers. The buffer can then be accessed directly. This method is used by PDM when there are 20 buffers or more.
- **RAM disk**. A record's RRN is used to directly locate the actual buffer in memory. This method is activated in PDM when the DBA assigns a *private* buffer to a single file with as many copies of buffers as logical blocks in a file. (A *private* buffer is a buffer exclusively assigned to a single database file.)

The following table shows algorithm use under different conditions (where *n* = the number of logical blocks in the file):

	Buffer search algorithms		
Number of buffers	**Serial scan**	**Hashing**	**RAM disk**
1 to 19	X		
20 to *n*-1		X	
> = *n*			X

As illustrated in the preceding table, the number of buffers is a determining factor in algorithm usage. Serial scanning is most efficient for a small number of copies of each buffer. As the number of buffers increases, performance can seriously degrade due to the CPU time required for each buffer search. In extreme cases, a search could require more time than a physical I/O, and thereby negate the advantage of introducing a large number of buffers.



As with any buffering product, PDM requires the memory buffers to be populated with data before they can be effective. The PDM fills a buffer when the data is first accessed from disk; buffers are not preloaded. Thus, there is no performance benefit for the first access to a SUPRA logical block. For example, if buffers are provided to buffer an entire data set in memory, and the data set is read with DATBAS RDNXT functions, performance will be no better than if a small number of buffers had been allocated. After the buffers have been loaded with data, subsequent reads should be much faster.

Optimizing data-set buffering

To take full advantage of PDM buffers, you must allocate the proper number of buffers to each data set. The proper number will vary by data set depending on the physical size of the data set and its activity. It will also vary by site depending on CPU speed, system load, and available memory. Small data sets and/or data sets with low I/O rates require fewer buffers. Larger data sets and/or data sets with high I/O rates require more buffers.

Before you can optimize your data-set buffers, you must collect statistics about your database activity, particularly the logical I/O (LIO) and physical I/O (PIO) rates for each data set. A physical read is a read that requires disk access, while a logical read without a physical read is a buffer hit (an in-memory read that does not require disk access). LIOs are good for performance; PIOs are bad for performance. Providing more buffers increases the likelihood of an LIO over a PIO.

I/O statistics are written to the PDM log file when a file is closed (if the PDM input parameter STATISTICS=Y). Statistics may also be gathered online by using the pdmstats.sh utility (see “pdmstats.sh” on page 237). However, before making your buffer allocation decisions, you must also consider the issue of PDM process memory.

Tuning PDM process memory

The more buffers you add in a dbmod, the more memory the PDM server will use. However, increasing buffers improves performance. Your goal is to balance performance and memory usage. Ideally, the PDM server process should have enough real memory to result in a zero page-fault rate (always finding the desired pages in memory).

Avoid trading a data set I/O for a virtual memory I/O. Even if the PDM server process has enough virtual memory to operate, you must also ensure that it has enough real memory to run at optimum efficiency. If the real memory is too small, the operating system will spend too much time page-faulting the PDM server process.

The memory that a particular dbmod is using can be determined after it is loaded by using the csiopcom command DISPLAY/DATABASES.

If you increase buffers in a dbmod, the PDM server process will require more shared memory to load the dbmod into memory. Insufficient shared memory in the PDM server process will result in a CSTI140F operator message, and an IDBM status will be returned to the application.

CONTROL: Manufacturing tuning considerations

In most shops, the MRP/MPS review processor (MPSMP070) is the longest-running batch job of the nightly job steam. Established CONTROL: customers have learned that a dbmod dedicated solely to MP070 is crucial for reducing MP070 run time. This is still true today, but the PDM buffer options can make MP070 run significantly faster. To optimize MP070 performance, consider the following:

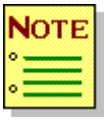
- Identify the subset of data sets used by MPSMP070. This may include (but not be limited to): BATD, BATM, BILL, INVD, LOCM, MSGS, OORD, ORDM, PART, PTDM, RQMT, SCAL, SCRM, SITD, SITM, SRCG, TABM, and TABV.
- Ensure that the LOGICAL-RECORDS-PER-BLOCK for each data set gives a SUPRA logical block size of 4096 bytes (possible exceptions are SCAL, PLV1, and /or LOCM which may need to be 8192, and should, therefore, not share a buffer pool with a 4096 data set).
- Identify the MP070 hot files that could use more buffer copies. This may include (but not be limited to): PART, PTDM, RQMT, ORDM, and OORD.
- Define more buffer copies for the batch environment to take advantage of additional free memory, which is typically at night.
- Use the csidba utility to create a heavily buffered database module.
- Run batch jobs using brunsl to take advantage of the single-task PDM (see “Single-task PDM” on page 117).
- Ensure that the read-ahead feature is active by defining the logical name CSI_READAHEAD with a value of YES (see the table under “Implementing logical names” on page 38).
- Turn off indexing at the start of the batch stream, and perform an index populate at the end of the batch stream.

Improving database performance with indices

You define indices during database definition by connecting them to a data set. Each data set can have one or more indices. Each index can contain one or more secondary keys, and each secondary key can be connected to one or more data items.

Consider the following when defining an index:

- For best performance, you should define no more than four secondary keys per index, and from one to four indices per data set.
- It is more efficient to group all the secondary keys for one data set in one index, rather than create several indices each containing one secondary key. However, if you want to be able to deactivate selected secondary keys, define each secondary key in its own index. When you deactivate an index, you implicitly deactivate all secondary keys it contains.
- If performance is a problem, you can make a small improvement by setting INDEX-READ-VERIFY=NO on the index attributes screen. This turns off the automatic check for index corruption and is fairly safe if used on an index you do not intend to deactivate. Frequent deactivation increases the risk of index records becoming outdated and no longer matching updated data records. If the index subsequently becomes corrupted and you have specified INDEX-READ-VERIFY=NO, results will be unpredictable.



You can obtain tremendous performance benefits from using indices during read-only operations; however, indexing becomes an overhead during maintenance operations such as INSERT, UPDATE, or DELETE.

Using secondary keys to access data offers, the following advantages over using primary keys and linkpaths:

- **Fast online lookup of nonvolatile data**. For example, customer records can be keyed on customer number. However, you could define an index on customer name and initials for more flexible retrieval of customer records. The flexibility derives from the fact that you can key into the customer record using name instead of number. Also, because secondary keys support generic reads (a search based on only part of the key), you can retrieve all records that match the partial key that is supplied. The use of secondary keys on records such as customer records, which seldom change once entered, incurs minimal performance overhead.

- **Periodic report generation based on secondary keys rather than primary keys**. For example, you may want to produce monthly reports ordered by product number within product class. The primary data set has product number as its primary key, and product class as a data field. To generate your reports, you define an index with secondary keys for product class and product number. You update the index and activate it once a month using the POPULATE function of the index utility. After you produce the report, you deactivate the index for the rest of the month to avoid the performance overhead of maintaining the index.

- **Reduced overhead in maintaining a sequenced primary linkpath**. By defining a secondary key on a sequencing field and the foreign key, the PDM no longer needs to maintain the chain of related records in sequence. Instead, it can use the index to retrieve records in order by inserting records at the bottom of the chain instead of sequentially searching the chain for the logical position. This may improve insert performance by avoiding the seat and reducing physical I/O. For example, to retrieve all identical surnames in first name order, place a secondary key on the sequencing field, which is the first name.

Designing application programs

Application program design can have a far-reaching effect on the efficiency of the system. The following guidelines apply to all programs that access the PDM.

Record holding

Record holding is a feature of the PDM that protects records from being updated by two or more tasks at the same time. The maximum number of records that can be held for a database is determined by the product of MAX-HELD-RECORDS and MAX-UPDATE-TASKS.

When the PDM input parameter MULTIHOLD=N, a task can explicitly read and hold only one record per file per logical unit of work. With each subsequent request to explicitly read and hold a record, all previous explicitly held records for the file are released for the task. With MULTIHOLD=Y, only the latest record explicitly read will be held for the task.

When the PDM input parameter MULTIHOLD=Y, a task can explicitly read and hold as many records as the record holding table permits.

All records held by a task will be released at the next COMMIT, RESET, or SIGNOFF.

Managing record holding

You can control the number of retries and the time between retries by setting the PDM input parameters RETRY= and INTERVAL= (see “Using a database prefix” on page 94). The PDM performs the number of retries specified in RETRY=, waiting for the length of time specified in INTERVAL= between retries before returning a HELD status to an application task. If you encounter an excessive number of HELDs, you can try increasing the number of retries and/or the interval between retries. More retries use up CPU cycles causing performance deterioration. A longer interval between retries reduces CPU cycles; however, the record may be stolen by another task before the interval is up. From the point of view of the system, it is better to increase the interval and reduce the number of retries.

Excessive record holding can cause a significant drop in performance. Some record holding is necessary to ensure data integrity. However, when another task attempts to access a held record, it must reset and retry until the record is released. These repeated application task retries increase processing time and impair performance.

There are three methods for minimizing record holding:

- **Optimizing the use of DML commands.** In a multitasking environment, records that are held by another task are temporarily unavailable. Because some records are accessed frequently by many programs, ensure that each record is held as infrequently and as briefly as possible. Do not hold a record until it is necessary to do so, and COMIT as soon as possible after updating. To this end, make sure that no program holds records while it waits for a response from the user. This allows other programs to access those records.

- **Reducing the duration of record holding.** Normally a program that needs to update several records places an exclusive lock on each record, collects the update information, updates each record, and then releases the locks. If the performance of your system is degraded by numerous processes trying to access the same records concurrently, you can reorganize your application programs so they gather all the information necessary to update the records first, and set an exclusive record lock only when all the update information has been collected. This reduces the amount of time each record is held by a process and helps minimize record contention.

- **Altering the order of record holding.** Sometimes many tasks compete to update a subset of data. You can reduce record contention in such cases by modifying the order in which your application programs issue requests for record locks. Ensure that your programs first request locks on the records that are more likely to be held by other tasks. Then, if another task already holds one of these records, the RESET requires less processing time.

Recovering from a deadly embrace

A deadly embrace can arise when two tasks need to update the same two records. To avoid a deadly embrace, specify a maximum number of retries that can occur if a required record is unavailable. When the program reaches this number of retries, reset the task to a known point in the transaction, releasing the record or records that it holds. You can set the maximum number of retries by setting the RETRY= parameter in the PDM input parameter file (see "Initiating the SUPRA Server Physical Data Manager" on page 81).

Optimizing the frequency of commits

Pay attention to the frequency of commits. Too many commits increase physical I/O and impact performance. Infrequent commits create long commit intervals, which lead to large record-holding tables that take time to search. Held records are unavailable to other users until the next COMIT, RESET, or SINOF function is performed.

Understanding read-ahead buffering

Read-ahead buffering is a mechanism used by the database access program (DATBAS) and PDM to improve performance of both sequential and serial reads when no record holding is being done. When your application executes a read function, RDNXT, READV, READR, or READX with the end parameter set to RLSE, the record is read and returned to your application without holding the record. It is also possible to turn off record holding by using the SINGLE SINON mode as explained in the *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240.

Read-ahead buffering is controlled by two parameters, READAHEAD_THRESHOLD1 and READAHEAD_THRESHOLD2. These parameters are set in the CSIPDMINP file (see "Creating a PDM input file" on page 96). These thresholds have default values of 10 and 100. This means that PDM will not begin to pre-read records until the application has made 10 requests for records from the same file with the same parameters. Once the first threshold has been reached and the application continues to perform READ functions for the same file with the same parameters, PDM will begin to fill the Readahead buffer with logical I/Os for RDNXT, READV, and READR until READAHEAD_THRESHOLD2 is reached. Once the second threshold is reached, PDM will begin to do physical I/O to fill the buffers. These parameters may be adjusted for specific applications and general performance tuning.

When DATBAS, either single-task or multitask, gets a request for a read function with no record holding, it requests the record from the PDM as usual. If the application makes additional requests without changing the file, the element list, the qualifier, the reference, or the end parameters, DATBAS requests that PDM place as many records as possible into the interprocess communication area. In the case of READX, DATBAS requests that 10 records be returned if they fit into the interprocess communication area.

Upon receiving the interprocess communication area from PDM, DATBAS copies it to one of three read-ahead buffers created automatically in the application’s local memory. There is a read-ahead buffer for RDNXT, one for READV and READR, and another for READX.

When the application makes another request for the same file, the record is extracted from the read-ahead buffer. No access to the PDM is required. This continues until all of the records in the buffer are exhausted or one of the parameters changes or a COMIT function is executed.

Because there are three read-ahead buffers, it is possible to intermix functions without losing the buffers. Your application could perform a mix of RDNXT, READV or READR, and READX as well as any other read or update functions and still retain all of the previously read records. Many programs perform sequences of reads to find data they are looking for.

As an example your program might do a RDNXT through a primary data set. For each primary record found, the program might perform a series of READV functions to read a set of related records. Both the RDNXT and READV functions in this example would be able to use the read-ahead buffers. If the program does a READV from two different data sets, alternating back and forth, no read-ahead buffering is possible for the READV functions. The performance of the application could be improved by accessing all of the records in one set, and then accessing all of the records in the other set. This process best uses read-ahead buffering.

Context position considerations

The position within a data set is retained by your application in the qualifier parameter. Your position is never lost regardless of the read-ahead buffering.

Application programming considerations

The application programmer does not have to specifically code for read-ahead buffers. The behavior of the PDM is identical with or without read-ahead buffering with the following exceptions:

- Because more records are being returned to the application program's process, there is a higher likelihood that the data read will be updated by another task while the reading application has the records in the read-ahead buffers. This could cause some applications to behave slightly differently with read-ahead buffering active. It may be more likely to get IVRP errors when reading with READV and READR when read-ahead is active.

- When read-ahead buffering is active, the behavior of *FILL= can be slightly modified. The values passed in the data area are not necessarily the ones used to fill the data area of the returned records. When DATBAS requests multiple records from PDM, the data area containing the *FILL= values are passed to PDM and are used to fill all of the records read. This works for most applications. However, this will not work for applications that modify the *FILL= data on each DATBAS call.

Turning off read-ahead buffering

If your application fits either of the above patterns, you will have to turn off the read-ahead feature. This can be done by defining the logical name CSI_READAHEAD with the following UNIX command:

```
csideflog -p CSI_READAHEAD NO
```

Turning on read-ahead buffering

To turn on the read-ahead feature, you may either remove the CSI_READAHEAD logical name with the csidellog command (see “Deleting logical names” on page 51) or change its value to YES with the csideflog command (see “Defining logical names” on page 45).

Printing read-ahead buffer statistics

Additionally, the logical name CSI_READAHEAD_STATISTICS may be set to the value of YES in order to produce read-ahead statistics. The statistics are printed in the file to which the CSIDAPLOG logical name points. If either the CSI_READAHEAD_STATISTICS or the CSIDAPLOG logical name is not defined, no statistics will be printed.

The following message will be printed in the CSIDAPLOG file:

CSTI436I

```
SINOF statistics (program name) (database)  Total Calls Buffered
Calls
RDNXT                                          (nnnn)      (nnnn)
READV/READR                                    (nnnn)      (nnnn)
READX                                          (nnnn)      (nnnn)
```

The Total Calls column shows the total number of calls made by the application for each function. The Buffered Calls column shows the number of calls that were executed using the read-ahead buffers. These functions were performed without accessing the PDM server.

The CSI_READAHEAD and CSI_READAHEAD_STATISTICS logical names can be defined in any logical name table. They do not have to be defined in the same table. By defining them in a process table, the values assigned will apply only to the process. By defining them in a group table, they will apply to all members of the group. By defining them to a system table, they will apply to all users of the PDM system.

Remote application considerations

Read-ahead buffering is supported for remote applications linked with DATBAS and using TCP/IP. The logical names CSI_READAHEAD, CSI_READAHEAD_STATISTICS, and CSIDAPLOG must be defined on the remote machine.

Performance tuning using the MAXDATA PDM input parameter

Read-ahead buffering uses the data area defined by the MAXDATA PDM input parameter to transfer records from PDM to DATBAS. The size of this area and the size of the data requested in the element list parameter limit the number of records that can be transferred in one request to the PDM server. Performance is usually improved by reducing the number of PDM server requests. By increasing the MAXDATA PDM input parameter, more records can be transferred from PDM to DATBAS in one PDM server request, reducing the number of PDM server requests required to read a data set or set of records. This is especially important if the amount of data being requested in your program is very large. The minimum and default value for MAXDATA is 4096, the maximum is 32767 (see “Creating a PDM input file” on page 96).

Controlling data item lists

It is preferable to minimize the length of the data item lists in your application programs. Do not request data items that the program does not process. Also, keep the sequence of the data item lists the same as the physical sequence of the data items in the record wherever possible. This speeds up processing and improves performance.

In addition, bind frequently accessed data items to increase efficiency. The program searches the data item list only once—the first time it is used. Data item binding is worthwhile whenever the same data item list is used more than once, regardless of the number of data items. Refer to the *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240, for further information about data item binding.

7

Using the UNIX online Help facility

The csihelp program is the online Help program for SUPRA Server under UNIX. It is a screen-based application that offers a multilevel Help system. Specific topics (subjects) may have successive numbers of subtopics, in which case the screen displays Help text in the form of an outline.

The Help program runs in both interactive and noninteractive mode. In interactive mode, you can move up and down through the Help text and obtain help on topics relative to your position in the system. You can move down through the Help text more than one level at a time by specifying a topic and successive subtopics. You can move up through the Help text one level at a time, and at each level the screen displays a list of subtopics. In noninteractive mode, you specify a topic that you need help with. The screen displays Help text for the topic you choose; then it allows you to immediately exit the Help program.

In both modes, you can perform generic searches, that may result in multiple matches, and you can turn on case sensitivity (the default is no case sensitivity).

Each Help file must have an associated Help index file that is built through the csimkhlp program. The Help index file has the same name as the Help file but is suffixed by ".hix". The program uses this index file to provide rapid access to the Help text for a particular topic. See "Using csimkhlp to create a Help file and Help index" on page 203 for details of the csimkhlp program and the format of the Help file.

Three logical names are used by the csihelp utility and the SUPRA PDM utilities that use the csihelp utility:

- CSIHELP is the full path and file name of the csihelp utility, which is used by the SUPRA PDM utilities to locate the csihelp binary file. For example:

```
csideflog -s CSIHELP /supra1/supra1_relx.x.x/bin/csihelp
```

- SUPRA_HELP is the full path to the subdirectory containing the SUPRA PDM Help files. For example:

```
csideflog -s SUPRA_HELP /supra1/supra1_relx.x.x//help
```

- CSI_HELPFILE is the full path and file name of the Help file to be used by csihelp, which is described in the following sections. Example:

```
csideflog -s CSI_HELPFILE
  /supra1/supra1_relx.x.x//help/csihelp.hlp
```

Entering the csihelp command

The format of the csihelp command is as follows:

csihelp [-h] [-i] [-s] [-t] [*help-file-path*] [*topic-list*]

-h

Description *Optional*. Specifies that the Help file to be used is to come from the command line.

Considerations

- If you specify -h, you must specify a Help file.
- If omitted, the Help file comes from the logical CSI_HELPFILE.

-i

Description *Optional*. Specifies that the program is to run in noninteractive mode.

Considerations

- This option is useful if the csihelp program is to be called from within a program.
- In noninteractive mode, the screen displays Help text for the topic selected on the command line. After the display, you immediately exit the program. In interactive mode, the screen prompts you to enter more topics.

-s

Description *Optional*. Turns on case sensitivity within the Help program.

Consideration If omitted, the program is not case sensitive.

-t

Description *Optional*. Specifies that the program prints an outline of topics under the specified topic.

Consideration The output is in the form of a nested topic list. For example:

```
TOPIC1
      TOPIC2
      TOPIC3
            TOPIC4
      TOPIC5
TOPIC6
```

[*help-file-path*]

Description *Conditional*. Required if you use -h. Specifies the path of the Help file to be used.

Consideration You can specify the path as a logical name.

[*topic-list*]

Description *Optional*. Specifies the Help topic to be displayed.

Format The output is in the form of a nested topic list. For example:

```
TOPIC1
      TOPIC2
      TOPIC3
            TOPIC4
      TOPIC5
 TOPIC6
```

Considerations

- If the topic is not at the top of the outline, you must specify all parents. You can perform generic searches by specifying the first *n* characters of a topic.
- If you do not use generic searching and interactive mode is on, you will be positioned at the specified topic once the help for that topic has been displayed. If you use generic searching and multiple matches are found, you are positioned at your last point in the outline.

Examples of the csihelp command

- The following example invokes interactive mode help with no case sensitivity and uses the Help file that logical CSI_HELPFILE points to. The screen displays an outline of Help topics.

```
$ csihelp
```

- The following example invokes interactive mode help with no case sensitivity using the file with the path: /users/mark/helpfil.hlp. The screen displays an outline of Help topics.

```
$ csihelp -h /users/mark/helpfil.hlp
```

- The following example invokes interactive mode help with no case sensitivity, and positions the help outline at the topic FRED using the Help file pointed to by the logical CSI_HELPFILE. The screen displays help for the topic FRED, and displays a list of subtopics for FRED.

```
$ csihelp  FRED
```

- The following example invokes noninteractive help with case sensitivity using the Help file pointed to by the logical CSI_HELPFILE. The screen displays Help for Mark, a subtopic of Steph, and control will be returned to the shell prompt.

```
$ csihelp -s -i  Steph Mark
```

- The following example invokes interactive mode help with no case sensitivity using the Help file pointed to by the logical CSI_HELPFILE. The screen displays help for any topics beginning with JI, which are subtopics of topics beginning with VA. If multiple matches are found, then the position is placed at the root of the outline. Otherwise, the position is placed at the only topic that met the search criterion.

```
$ csihelp  Va Ji
```

Using csimkhlp to create a Help file and Help index

The csihelp program uses an index file to provide rapid access to help on a particular topic. The index file itself is generated by the csimkhlp program. In order to generate an index, you must first create a Help file that has a fixed layout. The format of these files is as follows:

level-no. topic-name

help-text

level-no. topic-name

help-text

level-no. topic-name

help-text

level-no.

Description *Required*. Specifies the level of the particular topic. For instance, level 1 means that the topic is at the root of the help tree. Level 2 means that it is a subtopic of the last level 1 topic.

Format 1–3 numeric characters per level number

Considerations

- The level numbers can only increment by one, but they can jump back by any number. For example, the level numbers must form the tree structure. For example:

  The sequence 1,2,3,3,4,3,1 would be valid.

  The sequence 1,2,4,3,3,3,1 would be invalid.

- The first digit of the level number must appear in the first column of the Help file.

- You can have a maximum of 256 level numbers.

topic-name

Description *Required*. Specifies the name of the Help topic.

Format 1–80 alphanumeric characters

Consideration The topic name must appear on the same line as the level number and cannot span multiple lines (a line defined as a string of text separated by a new line character).

help-text

Description *Required*. Identifies the text for the Help topic.

Format You can have a maximum of 1024 alphanumeric characters in an entire Help entry. Therefore, the format for the Help text is 1 - (1024 - (*level-no.* + *topic-name*)). For example, if the level number is 3 characters and the topic name is 50 characters, the format for the Help text is 1–971.

Consideration The text can span multiple lines but it must not fall in the first column of any line because this is reserved for level numbers.

Once you create a Help file of this format, you can generate an index using the csimkhlp program. You can specify the name of the Help file on the command line, or the system will prompt you for the name when the program is run. The program creates a file with the same name as the original file but with '.hix' appended.

8

Database migration

This chapter provides details on how to move a database description and all data for that database. You can migrate any database that supports TOTAL or TOTAL-compatible DML. You must have:

- Data Definition Language (DDL) files describing your database
- Sequential files containing your data

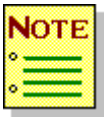
Before making any attempt to load DDL and/or any data into a database, back up your SUPRA Directory database (SUPRAD). This includes suprad.mod and all udd files, including logs for suprad. Make sure no one is using the Directory when you create the backup copy.

Use these basic migration steps on your source system:

1. Generate DDL file from source database.
2. Generate a sequential file containing the data from each data set.

Use these basic migration steps on your target system:

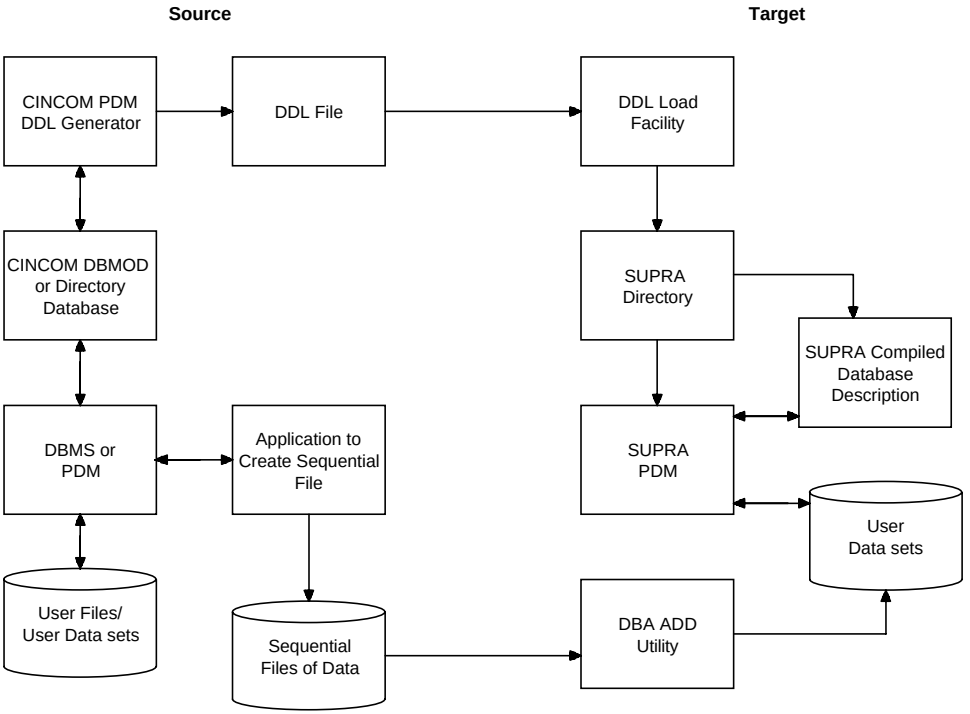
1. Back up the source SUPRA Server Dictionary and all associated data sets.
2. Load the DDL.
3. Validate and compile database description.
4. Format data sets.
5. Load data from the sequential files into the data sets.



You must write the application to copy the data in your source data sets into the sequential files that will be loaded into the target database. You may use an application development tool, such as MANTIS.

Migrating into SUPRA Server for UNIX

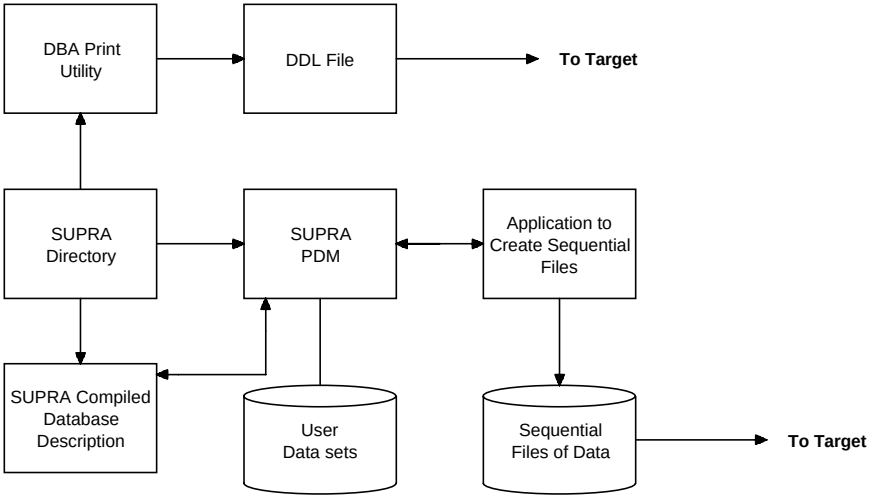
When migrating your database into SUPRA Server for UNIX, use the documentation from the source system to generate the DDL input file. You must generate sequential files of data, generating one from each of your data sets. Move these files to the new operating environment and do any required character set conversions before attempting the migration.



Migrating from SUPRA Server for UNIX

When migrating from SUPRA Server for UNIX, you must generate the DDL file and the sequential files of data. See “Generating a DDL file” on page 209 for details on generating the DDL file that will be used on the target system.

You must copy the data from each data set in your SUPRA Server database and put it into sequential files. These files will be used on the target system to add records into the new database. Character set conversion may be done either as you create the sequential files, or as you move these files to the target system. Operating environment restrictions may apply.



Generating a DDL file

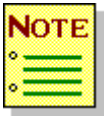
Before you can migrate your database to another operating environment or release of SUPRA Server, you must first generate a copy of your database definition in DDL format. To do this, you must set the logical definition CSI_DBA_PRINT; then select the DBA Print function. The logical name causes the DBA Print to generate a copy of the database description in DDL format.

The steps to generate a DDL file are:

1. Create a backup copy of the SUPRA Directory database (SUPRAD) and all associated files.
2. Define the logical CSI_DBA_PRINT to DDL as follows:

   ```
   csideflog -g CSI_DBA_PRINT DDL
   ```

3. Sign on to DBA and select Function 7, Print, from the Database Description Function menu to create a DDL file of the database you specify.



If you submit the print run to batch, remember the logical name available to the batch process. Defining the group logical name as shown above makes the print run available to the batch process; however, any other user in the same group that is printing a database description will also generate a DDL file instead of the normal print listing file.

Warning: When you generate the DDL file successfully, if you are migrating from VAX to SUPRA Server for UNIX, you must delete the following parameters from the DDL file:

```
GLOBAL-SECTION=

CALLING-MECH=

CLUSTER-SUPPORT=
```

Using the DDL Load Facility

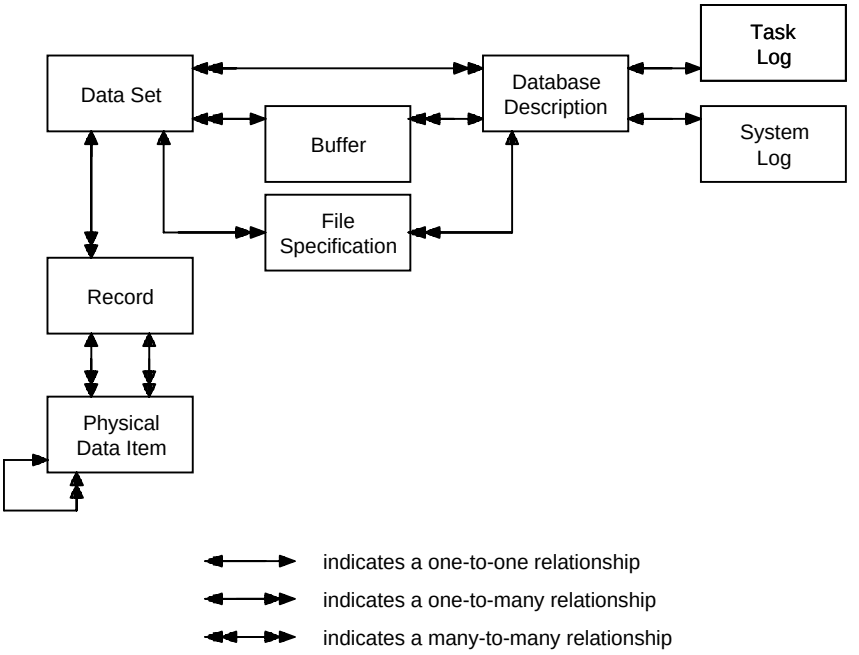
The DDL Load Utility, csiddlload utility, loads a database description onto the SUPRA Directory using an existing Data Definition Language (DDL) file. Use the csiddload utility to load a database description from one Directory to another. csiddlload is not intended to be a validation routine and assumes the DDL is correct.

The SUPRA Directory allows no duplication of database and data set names. If you are starting with a clean Directory, you will have no problems. However, if you already have databases and data sets defined, you must ensure that the name of the database and the names of the data sets you are about to add do not already exist on the Directory.

Be especially careful if the csiddlload utility has already been run unsuccessfully in update mode, as the Directory may have been partially updated. Delete the database descriptions through DBA before trying again, or restore the backup copy of the Directory, and use the restored version. If you attempt to load a database under a name that already exists on the Directory, SUPRA Server will respond with a message and return to command level without completing the load.

The csiddlload utility generates a listing output file named ddlload.log containing the DDL transactions and any errors (See “Checking csiddlload error conditions” on page 219 for details on error messages.) This file is created in your current default directory.

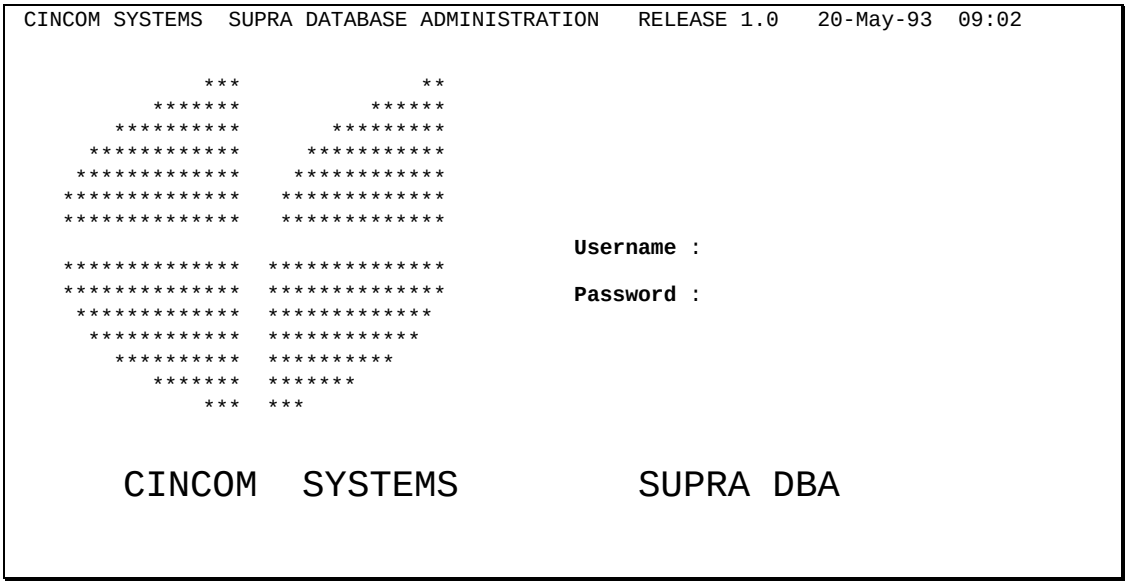
The following figure illustrates a data model of the physical Directory structure. The csiddlload utility adds the entities, their attributes, and the relationships according to this framework. Each box represents an entity and each line represents a relationship.



Signing on to csiddlload

Initiate the csiddlload utility from the command level to display the initial SUPRA Server sign-on screen:

```
csiddlload
```



The top line of the sign-on screen is a standard heading and is displayed on every screen requiring data entry. The information includes CINCOM SYSTEMS, the SUPRA Server title, the current date, and the time of entry.

Enter your valid username/password combination as defined to SUPRA Server. If unsuccessful after three attempts, the csiddlload utility automatically signs off.

Loading the DDL file

After you have successfully signed on, you will be prompted to give the full file specification for the DDL input file. Select CHECK ONLY mode or UPDATE mode, and specify the name of the database.

```
CINCOM SYSTEMS           DDL LOAD FUNCTION                  20-May-93   13:55

File containing DDL or <return> to exit
:testdb.ddl

CHECK ONLY or UPDATE data directory (C or U) :
(<PF4> will select U) : U

Do you want to take options for each data set in turn? (Y/N)
(<PF4> will select NO) : Y

Change Name of database ? (<PF4> will select TESTDB)
or <PF1> to exit :
```

File Containing DDL

Description Identifies the complete file specification of the DDL input file.

Format Up to 64 alphanumeric characters

Consideration If you enter the name of a file that does not exist or that cannot be opened, an error is displayed and you are allowed three more attempts. If these attempts fail, csiddlload terminates.

CHECK ONLY or UPDATE

Description Indicates whether you want to check or update the Directory.

Options

C Performs a syntax check and does not update the Directory.

U Updates the SUPRA Directory with your data.

Options for Each Data Set

Description Indicates whether or not you want to use the DDL Load options screen.

Options

Y Gives the DDL Load options screen for each data set specified in the DDL file.

N Each data set is automatically added and not renamed.

Consideration You should answer Y if you already have data sets in your SUPRA Directory that have the same name as the ones in your DDL file.

Change Name of database

Restriction Displayed only in update mode.

Description Specifies the new name of the database description as it will be stored in the SUPRA Directory.

Format 6 alphanumeric characters. The first character must be alphabetic.

Considerations

- To select the database name contained on the DDL file, press PF4.
- This parameter allows you to rename the database description before it goes to the Directory.
- If the name you select already exists on the Directory, csiddlload displays the message:

  ```
  (dbname) already exists on the directory <return>
  ```

  When you press RETURN, you exit to command level without completing the load.

After you enter the DDL Load Function screen, if you replied Y to the ‘options’ questions, SUPRA Server returns the DDL Load Options screen. Indicate what action you wish to take on the data set by entering the applicable number in the Option field.

```
CINCOM SYSTEMS        DDLLOAD LOAD FUNCTION - DATA SETS      20-May-93  13:56

        Options for data set SET1

           1: Insert data set as it is
           2: Rename and add this data set
           3: Ignore this data set

           Option:
```

Enter Choice No.

Description You are prompted to specify the action for each data set.

Options

1. Load the data set.
2. Load the data set under a different name (useful if a data set with that name already exists on the Directory).
3. Omit the data set from the load.

Considerations

- If you select option 2 and rename SET1 to SET2, as in the following example, the SUPRA Directory data set will be called SET2. All its records and data item names will be prefixed by SET2. The file specification names will not be changed. If required, use DBA functions to change them after the load is finished (refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260).
- If you select option 2 and rename a primary data set, check for linkpaths to related data sets that reference the original name. Change the linkpath references in the related data set using DBA functions.
- If you select option 3 to ignore this data set, it has the same effect as if all DDL for the data set had been removed from the input file. In the prologue, you may have specified a buffer intended for this data set. This buffer will be created on the SUPRA Directory, but it will not belong to any data set.

Example The following example shows the screens displayed if you select option 2 for Enter Choice No.

```
CINCOM SYSTEMS            DDLLOAD LOAD FUNCTION             20-May-93  13:57

        New name for data set SET1 : SET2
```

There is a time interval between each menu while the data set, records, data items, and file specifications are loaded onto the SUPRA Directory. The system then displays a confirmation screen.

```
CINCOM SYSTEMS            DDLLOAD RUN SUMMARY              20-May-93  13:57


        ddlload.log created for your information
        Number of DDL errors :   0
        Database TESTDB now exists on the directory









        << Press any key to terminate DDLLOAD >>
```

The entities and relationships are added to the Directory as they are read. Comments belonging to particular entities are added whether or not they occur in one contiguous block in the DDL file.

Checking csiddlload error conditions

The csiddlload utility uses the DDL input file and generates a listing output file. It is not a validation routine, and it will stop if an error is encountered on either of these files.

Status codes. If a bad status occurs on either the input file or the listing output file, the csiddlload utility displays a message indicating the name of the file and the error status code from the operating system. If csiddlload is attempting to open your DDL file, the error message is displayed after three unsuccessful attempts.

SUPRA Server error messages. SUPRA Directory errors, internal coding errors, and some I/O errors result in an error condition and the system displays a SUPRA Server error message. For the corrective action on any message you may receive, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022.

Listing output file messages. Error messages in the listing output file indicate an error in the DDL, which does not normally occur. However, warnings may be issued for DDL statements that are not valid for SUPRA Server for UNIX.

Sample DDL input file

The DDL input file contains the statements used to describe the database. The statements must be written in a fixed format. A sample file is presented below. For details on the specific values required for each statement, refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260. Statements in the DDL input file must conform to the following rules:

- All statements must begin in column 1. A blank terminates an entry. Anything following a blank is considered a comment. Thus, a comment can appear on the same line as a DDL statement. A blank in column 1 indicates a full comment line. Comments should be associated with the preceding statement. In the following example, the data item CUSTCTRL has two associated comments:

```
CUSTCTRL=6              CTRL field
 This is the key field
CUSTLKHD=8
```

- The INDEX= statement defines the fields that comprise the secondary key. This statement must be in the following format:

INDEX=*ffffppss=ffffdddd*(*x*),.....,END.

where: *ffff* = data set name

pp = primary index name

ss = secondary index name

dddd = data item name

(*x*) = data type

Each INDEX= statement may contain up to five data items per line. If the secondary key contains more than five data items, a continuation line is required. A continuation line must contain a hyphen (-) in column 1, followed by each data item (and its type), beginning in column 16. For example:

```
INDEX=CUSTP1S1=CUSTCTRL(C),CUSTDAT1(C),CUSTDAT2(C),
-             CUSTDAT4(C),CUSTDAT5(c),END.
```

- The primary index name must be in the format:

  INDEX-NAME=*ffff*IX*pp*

  where: *ffff* = data set name

  IX = constant denoting primary index

  pp = 2-character primary index name

- The secondary key name must be in the format:

  SECONDARY-KEY-NAME=*ffff*SK*ss*

  where: *ffff* = data set name

  SK = constant denoting secondary key name

  ss = 2-character secondary key name

Sample DDL input file

```
BEGIN-DATA-BASE-GENERATION
DATA-BASE-NAME=ORDERS
 This is a sample database
BEGIN-OPTIONS
PASSWORD=CINCOM
MAXIMUM-HELD-RECORDS=16
MAXIMUM-TASKS=10
MAXIMUM-UPDATE-TASKS=10
SHADOW=N
SINGLE-TASK=N
BYTE-ORDER=N
DATABASE-TYPE=G
BINARY-ZERO-KEY=N
NETWORK-SUPPORT=Y
TASK-LOG=Y
TASK-LOG-IO=5
FUNCTION-LOG=D
END-OPTIONS
SHARE-IO
```

```
IOAREA=PRIB=5           Buffer for primary data sets
IOAREA=RELB=5           Buffer for related data sets
END-IO
BEGIN-PRIMARY-DATA-SET
DATA-SET-NAME=CUST
  This is the CUSTOMER primary data set
IOAREA=PRIB
PRIMARY-DATA
CUSTROOT=8              ROOT field
CUSTCTRL=6              CTRL field
 This is the key field
CUSTLKHD=8
 This is the link to the ITEM data set
CUSTDATA=4
INDEX=CUSTC1C2=CUSTDATA(C),END.
INDEX=CUSTC1K1=CUSTCTRL(C),END.
INDEX=CUSTC2K2=CUSTDATA(C),CUSTCTRL(C),END.
END-DATA
TOTAL-LOGICAL-RECORDS=20008
LOGICAL-RECORD-LENGTH=26
LOGICAL-RECORDS-PER-BLOCK=61
ACCESS-MODE=Y
FILE-SPEC=CUST.QAR/SECTORS=12
START-INDEX-PHYSICAL-SPECS
INDEX-NAME=CUSTIXC1     Primary index
 This is a primary index
INDEX-CORRUPT-ACTION=0
INDEX-NULL-SORTING=H
INDEX-READ-VERIFY=Y
INDEX-FILE-SPEC=INDEX_DIR:cust1.idx
START-SECONDARY-KEY
SECONDARY-KEY-NAME=CUSTSKC2  Secondary key
 This index contains the DATA item
SECONDARY-KEY-UNIQUE=N
SECONDARY-KEY-DIRECTION=F
SECONDARY-KEY-POINTER-ORDERING=N
SECONDARY-KEY-POINTER-TYPE=D
SECONDARY-KEY-DATA-TYPE-SORTING=N
SECONDARY-KEY-DUPLICATES=5
```

```
END-SECONDARY-KEY
START-SECONDARY-KEY
SECONDARY-KEY-NAME=CUSTSKK1           Secondary key
 This index contains the CTRL item
SECONDARY-KEY-UNIQUE=H
SECONDARY-KEY-DIRECTION=Y
SECONDARY-KEY-POINTER-ORDERING=O
SECONDARY-KEY-POINTER-TYPE=D
SECONDARY-KEY-DATA-TYPE-SORTING=N
SECONDARY-KEY-DUPLICATES=5
END-SECONDARY-KEY
INDEX-NAME=CUSTIXC2                   Primary index
INDEX-CORRUPT-ACTION=O
INDEX-NULL-SORTING=H
INDEX-READ-VERIFY=Y
INDEX-FILE-SPEC=cust2.idx
START-SECONDARY-KEY
SECONDARY-KEY-NAME=CUSTSKK2           Secondary key
 This key contains the DATA and CTRL items
SECONDARY-KEY-UNIQUE=Y
SECONDARY-KEY-DIRECTION=F
SECONDARY-KEY-POINTER-ORDERING=N
SECONDARY-KEY-POINTER-TYPE=D
SECONDARY-KEY-DATA-TYPE-SORTING=N
SECONDARY-KEY-DUPLICATES=5
END-SECONDARY-KEY
END-INDEX-PHYSICAL-SPECS
END-PRIMARY-DATA-SET

BEGIN-RELATED-DATA-SET
DATA-SET-NAME=ITEM
IOAREA=RELB
RELATED-DATA
ITEMCODE=2
ITEMCORD=6
CORDLKOR=8=ITEMCORD
ITEMDATA=69
RECORD-CODE=HD                        Header record
.1.ITEMCUST=6                         Customer key
```

```
CUSTLKHD=8=ITEMCUST          Link to CUST primary data set
.1.ITEMHDDT=7
DATELKHD=8=ITEMHDDT
.1.ITEMHEAD=40
RECORD-CODE=IT
.1.ITEMITDT=7
DATELKIT=8=ITEMITDT
DATELKAL=8=ITEMITDT
.1.ITEMPART=6
PARTLKIT=8=ITEMPART
.1.ITEMDESP=6
DESPLKIT=8=ITEMDESP
.1.ITEMINV1=6
INVCLKIT=8=ITEMINV1
.1.ITEMQTYR=4
RECORD-CODE=DN
.1.ITEMDNDT=7
DATELKDN=8=ITEMDNDT
DATELKAL=8=ITEMDNDT
.1.ITEMDSP1=6
DESPLKDN=8=ITEMDSP1
.1.ITEMQTYD=4
.1.ITEMFILL=28
RECORD-CODE=IN
.1.ITEMINDT=00000007
DATELKIN=8=ITEMINDT
```

```
DATELKAL=8=ITEMINDT
.1.ITEMUNIT=5
.1.ITEMVATR=3
.1.ITEMFIL1=20
.1.ITEMINV2=6
INVCLKIT=8=ITEMINV2
.1.ITEMFIL2=4
END-DATA
TOTAL-LOGICAL-RECORDS=20196
LOGICAL-RECORD-LENGTH=85
LOGICAL-RECORDS-PER-BLOCK=12
LOAD-LIMIT=80                  Load limit
CONTROL-INTERVAL=204           Control interval
ACCESS MODE=Y
FILE-SPEC=ITEM.QAR/SECTORS=00000012
END-RELATED-DATA-SET


BEGIN-TASK-LOG-DATA-SET
LOG-BLOCKSIZE=1
LOG-BLOCKS=8000
FILE-SPEC=TLOG.QAR
END-TASK-LOG-DATA-SET


BEGIN-FUNCTION-LOG-DATA-SET
LOG-BLOCKSIZE=1
LOG-BLOCKS=500
FILE-SPEC=DUMPSLF_DIR:dumpslf_001
FILE-SPEC=DUMPSLF_DIR:dumpslf_002
END-FUNCTION-LOG-DATA-SET
END-DATA-BASE-GENERATION
```

Compiling database description

After you have successfully loaded your database description into the SUPRA Directory using the csiddlload utility, use the DBA Facilities (or csmcombat, the batch database compile program) to validate and compile your database description. (Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260.)

Formatting data sets

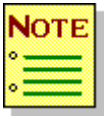
Before you can add records to the newly compiled database, you must format all files (data sets and recovery log files). Use the DBA Administration Facilities (or cstufmt) to format the files. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for details.

Adding records

Before you can use your new database, you must now load your data. As described in “Migrating into SUPRA Server for UNIX” on page 207, the data from your source system database was copied into sequential files that have been transferred to the target system. To add this data into your newly formatted data sets, use the Utilities option of the DBA Administrative Facilities. Refer to the *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220, for details. Alternatively, you may write an application to add the data to the database.

9

System Administration utilities



This is a new chapter and consists of information pertinent to this release.

The System Administration utilities provide information that can be used to help tune databases and PDM systems as well as diagnose problems. These utilities can be used in conjunction with the DBA utilities to provide a more complete and concise view of a database.

The following utilities are available:

- **csistats** Print statistics about databases and data sets
- **dbstat.sh** Print statistics about databases and data sets
- **pdmstats.sh** Print statistics gathered by a running PDM
- **csishoheld** Print a report showing the current held records
- **csidmpanl** Print a report showing PDM memory values saved in a dump analysis file by the PRINT command (csiopcom) or a fatal error

csistats

The database statistics utility csistats:

- Reports physical statistics about selected data sets
- Gathers physical database statistics for all data sets in a database

How the utility works

When you run this database statistic utility, it performs the following:

1. Signs on to the database to extract and display a list of all data sets defined in the database description file.
2. Extracts physical file attributes from the database description file for the selected data sets.
3. Opens each selected dataset file and scans through all records, gathering statistical information.
4. Prints a report of the statistical information, in:
   - Labeled format for a single data set.
   - Tabular format if all data sets were selected.

How to execute the utility

Execute the database statistics utility using the UNIX command csistats:

- Interactively from the UNIX shell.
- By executing a UNIX script containing the csistats command and input parameters (either online or in the background).

This utility prompts for a series of input parameters. The following lists the parameters and a description of each:

Prompt	Parameter	Result
`Enter database name:`	*database-name*	Enter the name of the database containing the data sets to be examined. The utility displays a list of all data sets defined for the database.
`Enter file name:`	*data-set-name* or ALL.	Enter either the name of a dataset to be examined or ALL. to examine all files in the database. After reporting on a single dataset, the utility prompts for another dataset name. When all data sets are selected, the utility returns directly to the shell after displaying the report.

You can press ENTER or EOF (usually CTRL-D) at any prompt to terminate the database statistics utility.

Shell execution

To execute the database statistics utility interactively, enter the command at the shell prompt and then respond to the prompts for database and data set names:

```
$ csistats
Enter database name: suprad
Enter file name: all.
```

Script execution

Create a UNIX script with any text editor. The database and dataset names are supplied as command parameters to the script. The script can then be executed either online or in the background.

Refer to the following for an example of creating and executing a script.

Example. The following is the UNIX script do_stats to generate statistics for a single dataset or all data sets in a database.

```
#
# do_stats script: executes Database Statistics utility
#                    $1 = database name
#                    $2 = dataset name or all.
#
if test “$1”=“”
then
        echo “No database name supplied”
else
if test “$2”=“”
then
        echo “No dataset name supplied”
else
csistats <<EOF
$1
$2
EOF
fi
fi
```

To execute the script online, enter:

```
$ chmod 744 do_stats
$ do_stats suprad all.
```

Execute the database statistics script as a background process by entering:

```
nohup do_stats suprad all. >stats.log 2>/dev/null &
```

This background process directs the report details (standard output) to a file named stats.log in the current directory and directs informational messages, prompts and dataset list (standard error) to the null (trashcan) file.

Example. The following is a sample screen displayed after running the csistats program online:

```
$ csistats
CSTU118I Cincom Database Statistics Utility version 2.0
Enter database name:  PERSON

DEPT  EMPL  WHST

Enter file name:  all.

CSTU105I Processing file /prod/SMOKE/directory/DEPT.PER
CSTU105I Processing file /prod/SMOKE/directory/EMPL.PER
CSTU105I Processing file /prod/SMOKE/directory/WHST.PER
Database statistics for PERSON run on:  05/05/97


           Total     Data      Empty     Control   Percent    Load    Full     File
Name       Records   Records   Records   Records   Full       Limit   Status   Status
DEPT (P)        10         0         9         1    10.00%                       ****
EMPL (P)        12         0        11         1     8.33%                       ****
WHST (R)       110         0       109         1     0.91%   80.00%              ****


$csistats
CSTU118I Cincom Database Statistics Utility version 2.0
Enter database name:  PERSON

DEPT EMPL WHST

Enter file name:  DEPT

CSTU105I Processing file /prod/SMOKE/directory/DEPT.PER
CSTU119I Statistics for dataset DEPT
CSTU120I Records Processed: 10
              File Status: ****
             Empty Records: 9
              Data Records: 0
            Control Records: 1
               Percent Full: 10.00%

Enter file name:
```

The following table lists descriptions of database statistics output fields for csistats. Field names may change or may not be reported depending on the file name selected.

Field name	Description
Name	The dataset name and type: (P) Primary (R) Related
Records Processed *or* Total Records	The total number of records formatted in the dataset.
Empty Records	The number of records currently free for user data.
Data Records	The number of records currently holding active user data.
Control Records	The number of records allocated as file and cylinder control records.
Percent Full	The number of (Data Records + Control Records) as a percentage of Total Records.
Load Limit	The load limit as defined in the database description file (related data sets only).
Full Status	Current file load indicator. For primary data sets, ♦ If load limit = 0.0, then percent full: * = 65% ** = 75% ***= 80% For related data sets: ♦ If load limit = 0.0, then percent full: * = 65% ** = 75% *** = 80% ♦ If load limit = 100.0, then percent full: % full + 15> load limit * % full + 5 > load limit ** % full > load limit ***
File Status	****, LOAD, FULL

dbstat.sh

The database statistics utility dbstat.sh:

- Reports physical statistics about all data sets
- Gathers physical database statistics for all data sets in a database

How the utility works

This utility is a script which calls two programs:

- **csidtlprt** Runs on the specified database name and displays the details of all the files
- **csistats** Called to gather the number of records in the file and the file status

The information is then extracted and printed in report format. The report prints to dbstat.# (in your working directory if you selected the spread sheet option) where # is the process ID number (pid).

Be sure you have write permission for the temporary files created in the /tmp directory.

How to execute the utility

Execute the database statistics utility using the command dbstat.sh database-name:

- Interactively from the UNIX shell
- By executing a UNIX script containing the dbstat.sh command and the command line parameters (either online or in the background)

Usage of the command is:

```
dbstat.sh [parameters] database-name
```

For example:

```
dbstat.sh -x burrys
```

The following table lists command-line parameters supported by dbstat.sh; keep in mind that these are case-sensitive:

Parameter	Description
- P password	Password
- s #	Column number to sort (default is 1)
- r #	Reverse sort order
- e “file1 file2 …”	File(s) to exclude from the output
- i “file1 file2 …”	File(s) to include for the output
- t #	Number of threads
- l #	Number of lines/page on report
- p	Format report for printer (66 lines/page)
- n	Display report without headers
- x	Spreadsheet setup (no headers, no totals, and comma separated fields)
- h/?	Help option lists available parameters to use

Example. The following is a sample screen displayed after running the dbstat.sh script online:

```
                                dbstat.sh Report
Database prefix:
Database name:  burrys

File  File  File  Alloc    Used     Pct   Alloc   File       Rcd   Rcd/  Blk  Rcds Load
Name  Type  Stat  Records  Records  Used  Blocks  Size (KB)  Lgth  Blk   Size  CI  Limt
--1-  --2-  --3-  ----4--  ----5--  --6-- ----7-- -----8---  --9-  -10-  -11- -12- -13-
BRAN   P    ****       78        0   0.0       3          9   115    26  3072    0    0
BRIN     R  ****      396        0   0.0       4         16    41    99  4096  198   79
BRMA     R  ****      124        0   0.0       2          5    41    62  2560   62   90
CUST   P    ****      120        0   0.0       3         10    89    40  3584    0    0
INVC   P    ****      210        0   0.0       5         10    48    42  2048    0    0
INVL     R  ****      525        0   0.0      15         22    43    35  1536  105   80
MANF   P    ****       78        0   0.0       2          3    39    39  1536    0    0
MANL     R  ****      138        0   0.0       6          6    44    23  1024   69   79
PGRP   P    ****       12        0   0.0       1          0    40    12   512    0    0
POLN     R  ****      260        0   0.0       4         12    47    65  3072   65   76
PORD   P    ****      140        0   0.0       5          5    36    28  1024    0    0
PROD   P    ****      126        0   0.0       7         14   111    18  2048    0    0
REGN   P    ****       16        0   0.0       1          0    31    16   512    0    0
STCK     R  ****     1776        0   0.0      24         84    48    74  3584  148   79
STRU     R  ****      444        0   0.0      12         18    41    37  1536  444   79
SUPP   P    ****       18        0   0.0       1          1    82    18  1536    0    0
VSNO   P    ****      236        0   0.0       4         12    52    59  3072    0    0
                    -------  -------              --------
                       4697        0                   227

Total files:  17,  Primary files:  10,  Related files:  7

Sort column:  1

start time Fri Jan 24 10:10:12 EST 1997
end   time Fri Jan 24 10:10:30 EST 1997
```

The following table lists descriptions of database statistics output fields for dbstat.sh:

Field name	Description
Database prefix	The prefix symbol.
Database name	The name of the database.
File Name	The dataset name.
File Type	The dataset type (P) Primary (R) Related
File Stat	The file status: ****, LOAD, FULL
Alloc Records	The number of allocated records. This column is totaled.
Used Records	The number of records currently holding data. This column is totaled.
Pct Used	The percentage used of the allocated records.
Alloc Blocks	The number of allocated blocks.
File Size (KB)	The file size in kilobytes. This column is totaled.
Rcd Lgth	The length of the record.
Rcd/Blk	The number of records per block.
Blk Size	The block size.
Rcd CI	The number of records per control interval.
Load Limt	The load limit as defined in the database description file (related data sets only).

pdmstats.sh

The PDM statistics utility pdmstats.sh provides statistic reports gathered from PDM statistics as shown in Appendix B:

- Reports PDM statistics from a snapshot
- Reports PDM statistics accumulated in PDM log files

How the utility works

There are a number of options when you run pdmstats.sh:

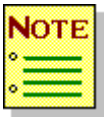
- Generate from a snapshot
- Generate from an accumulation of statistics from the PDM log file
- Generate for a specific database or for all databases
- Spreadsheet output prints to the files pdmstatdata.#, pdmstatfile.#, and pdmstattask.#, in your working directory, where # is the process ID number (pid)
- Formatting options for number of lines per page, remove zero-filled rows, sorting, and summarization

Be sure you have write permission for the temporary files created in the /tmp directory.

How to execute the utility

Execute the PDM statistics utility using the command pdmstats.sh:

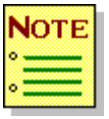
- Interactively from the UNIX shell
- By executing a UNIX script containing the pdmstats.sh command and the command line parameters (either online or in the background)



The default for pdmstats.sh is the snapshot of the current statistics.

Usage of the command is:

```
pdmstats.sh [parameters] [database-name]
```



database-name: ALL or a particular database name. If no database is specified, ALL is assumed.

The following table lists command line parameters supported by pdmstats.sh; keep in mind that these are case-sensitive:

Parameter	Description
- D db_name	Generates statistics from the PDM log file to which the CSIPDMLOG logical name points.
- l #	Number of lines/page on report.
- m db_name.ds_name	Summarization of a dataset in a database (only on table file).
- p	Format report for printer (66 lines/page).
- r	Remove zero-filled rows.
- s “table.# table.# …”	Sort according to statistics number and column number (`file.3` sorts table *file* at column *3*).
- x	Spreadsheet setup (no headers, no totals, and comma separated fields).
- h/?	Help option lists available parameters to use.

Based on the options selected, this utility produces a report on database statistics and file access, and possibly also task statistics.

Output type	Corresponds to the search of:
Data	♦ CSTI055S (database statistics) ♦ CSTI056S (database statistics)
File	♦ CSTI011S (DML file function statistics) ♦ CSTI012S (physical file I/O and miscellaneous file statistics)
Task	♦ CSTI008S (task sinof statistics)

Example. The following is a sample screen displayed after running the pdmstats.sh script with a database specified:

```
$ pdmstats.sh person
DBMOD Statistics for PERSON on Mon Jul 14 09:25:40 EDT 1997.

DATABASE STATISTICS
D_BASE TIME       SIGNONS SIGNOFFS DYNSINOFS    LFULS    DFULS CONTASKS CONFUNCS
  THREADS NFILFNCS TLFBUFSTLS CONUPTSKS
PERSON 09:25:40         1         1         0         0         0         1         1
       1         0          0         1

FILE STATISTICS
D_BASE DATASET  TIME     READS   WRITES     ADDS  DELETES     MISC    TOTAL   PR
EADS  PWRITES    LREADS  LWRITES BUFSNAVL BFLUSHES    HELDS   IHELDS
PERSON EMPL 09:25:40         0         0         0         0         0         0
  0         0         0         0         0         0         0         0
PERSON DEPT 09:25:40         0         0         0         0         0         0
  0         0         0         0         0         0         0         0
PERSON WHST 09:25:40         0         0         0         0         0         0
  0         0         0         0         0         0         0         0
TOTAL                 --------  --------  --------  --------  --------          ----
---  --------   --------  --------  --------  --------  --------  --------
                           0         0         0         0         0
  0         0          0         0         0         0         0         0
$
```

The following table lists descriptions of PDM statistics output fields for pdmstats.sh, the database statistics portion:

Field name	Description
D_base	Database name
Time	Time of statistics utility execution
Signons	Number of SINON DML functions issued for the database
Signoffs	Number of SINOF DML functions issued for the database
Dynsinofs	Number of dynamic SINOFS issued for the database
Lfuls	Reserved for future use
Dfuls	Reserved for future use
Contasks	Maximum number of tasks concurrently signed on to the database
Confuncs	Maximum number of functions concurrently issued to the database
Threads	Maximum number of threads concurrently active, executing a function on the database
Nfilfncs	Reserved for future use
Tlfbufstls	Number of times a task log file buffer had to be “stolen”
Conuptsks	Maximum number of update tasks concurrently signed on to the database

The following table lists descriptions of PDM statistics output fields for pdmstats.sh, the file access portion:

Statistics output	Description
D_base	Database name
Dataset	Data set of the particular database
Time	Time of statistics utility execution
Reads	Total number of READ DML functions used on the file since it was physically opened
Writes	Total number of WRITE DML functions used on the file since it was physically opened
Adds	Total number of ADD DML functions used on the file since it was physically opened
Deletes	Total number of DELETE DML functions used on the file since it was physically opened
Misc	Total number of RQLOC DML functions used on the file since it was physically opened
Total	Total number of the statistics (Reads, Writes, Adds, Deletes, and Misc) per each data set
Preads	Number of physical reads performed on the file
Pwrites	Number of physical writes performed on the file
Lreads	Number of logical read operations performed on the file
Lwrites	Number of logical write operations performed on the file
Bufsnavl	Number of times a logical read function had to wait for a buffer to be freed
Bflushes	Number of times a logical read function had to flush a buffer in order to use it
Helds	Number of times any function issued for this file had to be retried because of a record hold
Ihelds	Number of internal held conditions encountered

The following table lists descriptions of PDM statistics output fields for pdmstats.sh, the task statistics portion:

You must select STATISTICS=Y in the PDM input file to generate this output.

Statistics output	Description
D-base	Database name
Time	Time of statistics utility execution
Reads	Total number of READ DML functions issued by the task
Writes	Total number of WRITE DML functions issued by the task
Adds	Total number of ADD DML functions issued by the task
Deletes	Total number of DELETE DML functions issued by the task
Misc	Total number of COMMIT, RESET, SINON, and SINOF DML functions issued by the task
Helds	Number of times any function issued by this task had to be retried because of a record hold
Ihelds	Number of internal held conditions encountered

csishoheld

The held record utility csishoheld:

- Reports the held records for a specified database or all databases
- Reports the held records for specified files
- Reports the held records for specified tasks

How the utility works

The utility is a script performing the following steps:

- Defines the logical name CSI_DMPANL
- Executes the PRINT command of csiopcom
- Executes the utility csidmpanl that produces an output file
- The output file is analyzed by csishoheld, and a report is produced

How to execute the utility

Execute the held record utility using the command csishoheld:

- Interactively from the UNIX shell
- By executing a UNIX script containing the csishoheld command and the command line parameters (either online or in the background)

Usage of the command is:

```
csishoheld [parameters] database-name
```

For example:

```
csishoheld burrys
```

The following table lists command line parameters supported by csishoheld; keep in mind that these are case-sensitive:

Parameter	Description
-f filelist	Restrict listing to data about data sets whose names are given in filelist
-p proclist	Restrict listing to data about processes whose process ID numbers are given in proclist
-h	Print help information to standard output

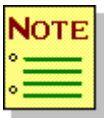
csishoheld prints information from the record holding table for a database. If no options are specified, information for all currently held records is printed. The output consists of process details, data set name, and RRN.

The database-name argument specifies the database whose record holding table is to be examined. The information displayed can be controlled by the selection of options. Options using lists as arguments can have the list specified in one of two forms:

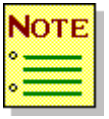
- A list of identifiers separated from one another by a comma
- A list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces

Example. The following is a sample output after running csishoheld with no options:

```
PID             USER            TID             TASKNAME            COMMAND
9171            csi             0005            TESTER01            csichckr
  DATASET:      EMPL
  RRNs HELD:    00000001
```



Things can change while csishoheld is running; the picture it gives is only a snapshot in time.



Write access to the current working directory is required as csishoheld creates temporary files in this location.

csidmpanl

The dump analysis utility csidmpanl:

- Displays information that is stored in memory assigned to a database
- Can be executed stand-alone, or within the csishoheld script

How the utility works

This utility requires the following steps be performed:

1. The logical name CSI_DMPANL must be defined to PDM using the csideflog function. This logical name must specify a full pathname for the file that will contain the dumped information.

2. When the snapshot is to be taken, run csiopcom against an active database and specify the PRINT option. If the logical name CSI_DMPANL is not defined, the file “CSI_DMPANL” will be created in the directory from which the PDM was started.

   The file specified by the logical name CSI_DMPANL will be generated. This file can also be generated by a FATAL PDM error. csidmpanl can be used to analyze the file from either source.

3. csidmpanl is an interactive utility prompting the user for an output file name. Following this, the user will respond with Y or N to 12 tables to be dumped.

How to execute the utility

Consider the following:

- Execute the dump utility by using the command csidmpanl
- Usage of the command is:

```
csidmpanl
```

- You will be prompted for an output file name, along with prompts for each table that is to be dumped

The following table lists the 12 tables that may be dumped:

Table	Description
TTE	Task Table for each TASK
TTC	Thread Context
TDP	DBMOD
TGF	File Table
TEB	Extent Block
TEL	Element Table for each File
TVL	Linkpath Table for each File
Buffers	Buffers for each file
TRH	Record Holding Table
TSC	Schema for each Task
TLM	Task Log allocation map
TLL	Task Log block group chain

csidmpanl will search the file pointed to by the logical name CSI_DMPANL for information about the tables requested. The formatted output will be placed in the output file name. The report will contain internal PDM information for each table. This will contain the address in memory of each table, with addresses of areas within each table. This information is used by Cincom Support to analyze a PDM system. In addition, the csishoheld utility uses the information to prepare a formatted report on held records.

A

Example user exits

This appendix presents example user exits written in COBOL and FORTRAN. The COBOL example uses both the before and after exits; the FORTRAN example uses only the after exit. Command files for compiling and linking both examples are provided at the end of each section.

COBOL user exits

The COBOL user exits trace DML parameters to check that they are being passed correctly between the application program and the PDM. The before exit displays the parameters before the call to the PDM. The after exit displays the parameters after the call to the PDM. In addition, the after exit shows the status of each DML call. The traced DML functions are ADDVA, ADDVB, ADDVC, DELVD, READR, and READV.

COBOL user exit 1

```
Identification division.
Program-id. CSI_INTUSEREX1.
*
* This Before exit traces calls to most
* Related DML functions to SUPRA User database files.
*
Data Division.
*
Working-storage Section.
*
01 i            Pic s9(4) Comp.
*
01 key-dec.
  03 key-dec-1 Pic 9(8).
  03 key-dec-2 Pic 9(8).
*
01 key-character.
  03 key-char   Pic x Occurs 8.
*
01 refer.
  03 refer-char Pic x Occurs 4.
*
Linkage Section.
*
01 function     Pic x(5).
01 stat         Pic x(4).
01 data set.
  03 udd        Pic x(3).
  03 filler     Pic x.
01 param-4.
  03 master-key-1                      Pic s9(9) Comp.
  03 master-key-2                      Pic s9(9) Comp.
01 linkpath     Pic x(8).
01 param-6.
  03 data-1     Pic s9(9) Comp.
  03 data-2     Pic s9(9) Comp.
Procedure Division Using function, stat, data
  set,param-4,linkpath,param-6.
s1 Section.
*
s.
* Skip calls to DIR DB
If udd = ''UDD'' Go To b.
* Test function type
   If function = ''ADDVA'' Or ''ADDVB'' Or ''ADDVC'' Or
      ''DELVD'' Or ''READR'' Or ''READV''
     Go To rela-set.
*
* other function
*
   Go To b.
*
rela-set.
*
* make key parameter hexadecimal for first 8 bytes.
   Move data1 to key-dec-1
   Move data2 to key-dec-2*
   Move param-6 To key-character.
   Perform convert-refer.
   Move master-key-1 To refer.
   Perform convert-hex-refer.
```

```
   Display ''B '' ,function,'' '',data set,''      Ref:'',
     master-key-1 conversion,''='', refer, '' '',linkpath,'' Key
  1-8:''
     key-dec-1, '' '',key-dec-2, ''='',key-character. text

*
b.
   Exit Program.
*
convert-key Section.
s.
   Perform do-it Test Before Varying i From 1 By 1 until i > 8.
   Go To e.
do-it.
   If key-char(i) Not Numeric And key-char(i) Not Alphabetic
     Move ''.'' To key-char(i).
e.
   Exit.
*
convert-key-refer Section.
s.
   Perform do-it Test Before Varying i From 1 By 1 until i > 4.
   Go To e.
do-it.
   If refer-char(i) Not Numeric And refer-char(i) Not Alphabetic
     Move ''.'' To refer-char(i).
e.
   Exit.
```

COBOL user exit 2

```
Identification division.
Program-id. CSI_INTUSEREX2.
*
* This After exit traces calls to most
*      Related DML functions to SUPRA User database files.
*
Data Division.
*
Working-storage Section.
*
01 i            Pic s9(4) Comp.
*
01 key-dec.
  03 key-dec-1 Pic 9(8).
  03 key-dec-2 Pic 9(8).
*
01 key-character.
  03 key-char   Pic x Occurs 8.
*
01 refer.
  03 refer-char Pic x Occurs 4.
*
Linkage Section.
*
01 function     Pic x(5).
01 stat         Pic x(4).
01 data set.
  03 udd        Pic x(3).
  03 filler     Pic x.
01 param-4.
  03 master-key-1                      Pic s9(9) Comp.
```

```
  03 master-key-2                        Pic s9(9) Comp.
01 linkpath     Pic x(8).
01 param-6.
  03 data-1     Pic s9(9) Comp.
  03 data-2     Pic s9(9) Comp.
/
Procedure Division Using function, stat, data
  set,param-4,linkpath,param-6.
s1 Section.
*
s.
* Skip calls to DIR DB
If udd = ''UDD'' Go To b.
* Test function type
   If function = ''ADDVA'' Or ''ADDVB'' Or ''ADDVC'' Or
      ''DELVD'' Or ''READR'' Or ''READV''
     Go To rela-set.
*
* other function
*
   Go To b.
*
rela-set.
*
* make key parameter hexadecimal for first 8 bytes.
*
   Move data1 to key-dec-1
   Move data2 to key-dec-2
   Move param-6 To key-character.
   Perform convert-hex.
   Move master-key-1 To refer.
   Perform convert-hex-refer.
   Display ''A '' ,function,'' '',data set,'' '',stat,'' Ref:'',
     master-key-1 conversion,''='', refer, '' '',linkpath,'' Key
  1-8:''
     key-hex-1, '' '',key-hex-2, ''='',key-character. text
*
b.
   Exit Program.
*
convert-key Section.
s.
   Perform do-it Test Before Varying i From 1 By 1 Until i > 8.
   Go To e.
do-it.
   If key-char(i) Not Numeric And key-char(i) Not Alphabetic
     Move ''.'' To key-char(i).
e.
   Exit.
*
convert-refer Section.
s.
   Perform do-it Test Before Varying i From 1 By 1 Until i > 4.
   Go To e.
do-it.
   If refer-char(i) Not Numeric And refer-char(i) Not Alphabetic
     Move ''.'' To refer-char(i).
e.
   Exit.
```

COBOL command file to compile and link the exits

```
$!
$! Command file to compile and link the sample COBOL PDM User
  Exit.
$!
$! Compile user exit 1 or Before Exit
$Cobol/noansi csi_pdm_exit1
$!
$! Compile User exit 2 or After Exit.
$Cobol/noansi csi_pdm_exit2
$!
$! Link the two compiled objects to form an Executable
$! Image. Note that the names of the exits, CSD_UPDM_USEREX1
$! and CSD_UPDM_USEREX2 MUST be declared as UNIVERSAL
$! symbols.
$!
$! To activate the User exits, do the following before
$! running your SUPRAPDM Applications :
$!  $DEFINE CSI_USEREX dev:_[directory_]CSI_PDM_EXIT.EXE
$!
$Link/share=csi_pdm_exit.exe csi_pdm_exit1, csi_pdm_exit2, -
sys$input/option
universal=CSD_UPDM_USEREX1,CSD_UPDM_USEREX2
```

FORTRAN user exit

The FORTRAN user exit traces the primary and related user database functions to check that the DML parameters are passed correctly between the application program and the PDM. As this example shows, you do not need to use both before and after exits.

FORTRAN user exit

```
      subroutine csd_updm_userex2(p1,p2,p3,p4,p5,p6)
c
c   SUPRAPDM After Exit example :
c   This sample Fortran program traces some of the parameters
c   after making a DML call to the user database files.
c
c
c   Note that this exit uses Numeric dummy arguments
c   because they are passed by reference from the PDM.
c   Dummy Numeric arguments may not be Equivalenced to character
c   strings, thus the mapping of the arguments from numerical
c   addresses to character strings are done in two steps.
c
      implicit none
      character*5 function
      character*4 stat, data set
      character*8 linkpath
      real*8 p1,pp1
      integer*4 p2,p3,pp2,pp3
      real*8 p4,p5,pp5,p6
      equivalence (pp1,function),(pp2,stat)
      equivalence (pp3,data set),(pp5,linkpath)
c
c
      pp1=p1
      pp2=p2
      pp3=p3
```

```
c
c    Ensure we only trace calls to user databases by examining
  the
c    data set argument - this parameter is only available if it
  is
c    one of a data set access functions.
c
    if ( ((function .eq. 'ADD-M' ) .or.
1         (function .eq. 'DEL-M') .or.
2         (function .eq. 'READM') .or.
3         (function .eq. 'WRITM' )) .and.
4         ( data set .ne. 'UDD1' ) .and.
5         ( data set .ne. 'UDD2' ) .and.
6         ( data set .ne. 'UDD3' ) ) goto 20
c
    if ( ((function .eq. 'ADDVA') .or.
1         (function .eq. 'ADDVB') .or.
2         (function .eq. 'ADDVC') .or.
3         (function .eq. 'ADDVR') .or.
4         (function .eq. 'DELVD') .or.
5         (function .eq. 'READD') .or.
6         (function .eq. 'READR') .or.
7         (function .eq. 'READV') .or.
8         (function .eq. 'WRITV')) .and.
9         ( data set .ne. 'UDD1' ) .and.
1         ( data set .ne. 'UDD2' ) .and.
2         ( data set .ne. 'UDD3' ) ) goto 30
c
c   Ignore the rest of the functions
      goto 99
c
20      write (*,100) function,stat,data set,p4
      goto 99
30      pp5=p5
      write (*,110) function,stat,data set,linkpath,p6
99      return
c
100     format (' ',A5,' ',A4,' ',A4,' ',Z16)
110     format (' ',A5,' ',A4,' ',A4,' ',A8,' ',Z16)
      end
```

FORTRAN command file to compile and link the exit

```
$!
$!  Command file to compile and link the sample FORTRAN PDM User
  Exit
$!
$!  Please note that this example uses only the After exit
$!
$! Compile the Fortran program containing the After Exit.
$ FOR CSI_PDM_EXIT_FOR
$!
$! Link the compile object to form an Executable Image.
$! Note that the name of the exit, CSD UPDM EXIT2 MUST be
  declared
$! as UNIVERSAL Symbol.
$!
$! To activate the User Exit, do the following before running
  your
$! SUPRAPDM applications :
$!
$!       $DEFINE CSI_USEREX_dev:_[directory_]CSI_PDM_EXIT_FOR.EXE
$!
$! To deactivate the exit, DEASSIGN the logical name CSI_USEREX.
$Link/share=CSI_PDM_EXIT FOR.EXE
  CSI_PDM_EXIT_FOR.OBJ,sys$input/opt
UNIVERSAL=CSD_UPDM_USEREX2
GSMATCH=ALWAYS,0,0
$!
```

B

PDM statistics output

This appendix describes the statistics written to the log file when you select STATISTICS=Y in the PDM input file.

Task statistics

The PDM produces this message whenever a task signs off:

```
CSTI008S (database), (process-name), (task-id), TASK SINOF
STATISTIC
READS   WRITES   ADDS    DELETES   MISC    HELDS    IHELDS
```

(*reads*) (*writes*) (*adds*) (*deletes*) (*misc*) (*helds*) (*int helds*)

The columns have the following meaning:

READS	The total number of READ DML functions issued by the task.
WRITES	The total number of WRITE DML functions issued by the task.
ADDS	The total number of ADD DML functions issued by the task.
DELETES	The total number of DELETE DML functions issued by the task.
MISC	The total number of COMIT, RESET, SINON, and SINOF DML functions issued by the task.
HELDS	The number of times any function issued by this task had to be retried because of a record hold. For example, if a function attempts to access a record that is already held for update, that function is backed out, a hold request is added to the queue for that record and the function is placed in the retrying queue. When the record is free, the function is transferred to the allocating queue ready to be restarted.
IHELDS	The number of internal held conditions encountered. In practice, this value is the same as the HELDS column except that in this case the function was able to restart immediately instead of waiting in the retrying queue.

File statistics

The PDM produces this message whenever it physically closes a file. A file is physically closed when the last task to use it issues a logical close for that file. This could be as the last task signs off during normal processing or as a result of an operator command to close down the database or the PDM (SHUTDOWN, UNLOAD).

```
CSTI011S (database), (data-set), DML FILE FUNCTION STATISTICS,
READS     WRITES    ADDS      DELETES     MISC
(reads)   (writes)  (adds)    (deletes)   (misc)
```

The columns have the following meaning:

READS — The total number of READ DML functions used on the file since it was physically opened.

WRITES — The total number of WRITE DML functions used on the file since it was physically opened.

ADDS — The total number of ADD DML functions used on the file since it was physically opened.

DELETES — The total number of DELETE DML functions used on the file since it was physically opened.

MISC — The total number of RQLOC DML functions used on the file since it was physically opened.

```
CSTI012S (database), (data-set), PHYSICAL FILE I/O & MISC FILE
  STATISTICS,
  BUFSNAVL        BFLUSHES        PREADS          PWRITES
  (buf not avail) (buf flushes)   (phys reads)    (phys writes)

  LREADS          LWRITES         HELDS           IHELDS
  (logical reads) (logical writes) (helds)        (int helds)
```

The columns have the following meaning:

BUFSNAVL — The number of times a logical read function had to wait for a buffer to be freed.

BFLUSHES — The number of times a logical read function had to flush a buffer in order to use it. A read function first searches the pool of buffers for the record. If the record is not there, the read function searches for an empty buffer to use. If no empty buffer is available, the read function searches for an unlocked buffer. A BFLUSH occurs when an unlocked buffer is found that contains modified records that must be flushed before the buffer can be used.

PREADS — The number of physical reads performed on the file. A physical read reads in as much of the file as can be contained in one buffer. In VMS, for example, 25 physical reads would be needed to read a file that has 50 VMS sectors and a buffer size of 1024 bytes. If a file has too few buffers, or if the buffers are too small, then the number of physical reads needed increases.

PWRITES — The number of physical writes performed on the file. Normally, this value equals the number of BFLUSHES plus the number of buffers.

LREADS — The number of logical read operations performed on the file. The target of a logical read is one database record that may be found in one of the buffers for the file. A physical read is performed if the database record is not found in a buffer. The number of logical reads should, therefore, greatly exceed the number of physical reads. The number of logical reads should also exceed the number of DML functions since, for example, one ADDVR could generate three or four logical reads.

LWRITES	The number of logical writes used on the file. Currently, this column gives useful figures only for the task log and system log.
HELDS	The number of times any function issued for this file had to be retried because of a record hold. For example, if a function attempts to access a record that is already held for update, that function is backed out, a hold request is added to the queue for that record, and the function is placed in the retrying queue. When the record is free, the function is transferred to the allocating queue ready to be restarted.
IHELDS	The number of internally held conditions encountered. In practice, this value is the same as the HELDS column except that, in this case, the function restarted immediately instead of waiting in the retrying queue.

Database statistics

The PDM produces this message when the last task signs off from the specified database.

```
CSTI055S (database), DATABASE STATISTICS,
SINONS     SINOFS     DYNSINOFS          LFULS     DFULS
(sinons)   (sinofs)   (dynamic sinofs)   (lfuls)   (dfuls)
```

The columns have the following meaning:

SINONS	The number of SINON DML functions issued for the database.
SINOFS	The number of SINOF DML functions issued for the database.
DYNSINOFS	The number of dynamic SINOFS issued for the database. Dynamic SINOFS include applications that exit without signing off or applications that are signed off as a result of an operator command.
LFULS	Reserved for future use.
DFULS	Reserved for future use.

```
CSTI056S (database), DATABASE STATISTICS,
CONTASKS   CONFUNCS   THREADS   NFILFNCS   TLFBUFSTLS   CONUPTSKS
nnnnnn     nnnnnn     nnn       nnnnnn     nnnnnn       nnnnnn
```

The columns have the following meaning:

CONTASKS The maximum number of tasks concurrently signed on to the database.

CONFUNCS The maximum number of functions concurrently issued to the database. Since a task may issue only one function at a time, the CONFUNCS figure should be less than or equal to the CONTASKS figure.

THREADS The maximum number of threads concurrently active, executing a function on the database.

NFILFNCS Reserved for future use.

TLFBUFSTLS The number of times a Task Log File (TLF) buffer had to be “stolen.” When a thread attempts to acquire a TLF buffer, it first tries to get a free buffer. If none are available, the task must steal a buffer that is not currently locked after first flushing it.

CONUPTSKS The maximum number of update tasks concurrently signed on to the database.

C

Example mailbox read program

This appendix presents a C program to read the PDM messages from a named pipe. In this example, the mailbox is SUPRA1-000114, constructed using the equivalence name for CSI_PDMID (SUPRA1 in this case) and the group in which the PDM is running (000114 in this case). Substitute your own mailbox name to use this program.

Before you can read PDM messages from a mailbox you must do the following:

- Set the PDM input file parameter MRELAY=Y to send all PDM messages from CSIPDM to your mailbox
- Define the logical name CSI_MRELAY to send all console messages from CSIDAP to your mailbox

See “Writing your own interface to SUPRA Server PDM” on page 154 for a detailed description of how to write a user interface to SUPRA Server PDM.

Example mailbox read program, MAIL-BOX-TRAP.C

```
/*************************************************************/

/*                      MESSAGE_READ                         */

/*                                                           */

/* Sample program to read messages sent by the PDM to a named  */

/*   pipe                                                    */

/*************************************************************/


#include <stdio.h>

main ()

{
char  pipe_name[] = "SUPRA1_000114";  /* Name of pipe          */

int   pipe_fd;                        /* Pipe file descriptor  */
char  message_buf[1024];               /* Buffer for message    */

/* Open the named pipe */

if ( pipe_fd== -1 )
{
  printf ( "\nCannot open pipe %s, pipe_name ) ;
  return;
}
else
  /* Keep reading data */
  while ( read (pipe_fd, message_buf, 1024 ) != -1 )
      printf ( "%s", message_buf ) ;

return;

}
```

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W